

Graphics Library Package

Software Manual

PMC and PCI Graphics

PRELIMINARY

1 July 1999

Peritek Corporation
5550 Redwood Road
Oakland, CA 94619

(510) 531-6500
FAX (510) 530-8563
email: support@peritek.com
web: <http://www.peritek.com>

Notices

Information contained in this manual is disclosed in confidence and may not be duplicated in full or in part by any person without prior approval of Peritek Corporation. Its sole purpose is to provide detailed documentation to effectively install and operate Peritek equipment. The use of this document for any other purpose is specifically prohibited.

The information in this document is subject to change without notice. The specifications of **GLP** and other components described in this manual are subject to change without notice. Peritek Corporation assumes no responsibility for any errors or omissions that may occur in this manual.

Peritek Corporation assumes no responsibility for the use or reliability of software or hardware that is not supplied by Peritek, or which has not been installed in accordance with this manual.

GLP and **Peritek** are trademarks of Peritek Corporation. LynxOS, HP-UX, OS-9, pSOSystem, Solaris, SunOS, OSF/1, Ultrix, VMEexec, V/68 SVR3, V/88 SVR3, TURBOchannel and VxWorks are registered trademarks of Lynx, Hewlett-Packard, Microware, Integrated Systems, Sun, Sun, Digital Equipment Corporation (DEC), DEC, Motorola, Motorola, DEC, and Wind River, respectively. OpenGL is a registered trademark of Silicon Graphics, Inc.

Copyright © 1998 by Peritek Corporation

Table of Contents

1	Introduction.....	8
1.1	Overview of the Graphics Library Package (GLP)	8
1.2	Installation.....	9
1.3	Getting started.....	10
1.4	Using this manual.....	11
2	The libWS library.....	12
2.1	Data Structures.....	12
Ws_device	12	
Ws_workstation.....	12	
Ws_region.....	12	
Ws_video_timing.....	12	
Ws_event.....	12	
2.2	Functions.....	12
ws_registered_devices.....	12	
ws_configured_workstations.....	13	
ws_configured_devices	13	
ws_open_workstation.....	13	
ws_close_workstation.....	13	
ws_get_video_timing	13	
ws_set_video_timing.....	14	
ws_get_regioninfo.....	14	
ws_map_region.....	14	
ws_get_colormap_random.....	14	
ws_set_colormap_random.....	14	
ws_get_colormap	14	
ws_set_colormap.....	14	
ws_associate_cursor.....	14	
ws_get_cursorinfo	14	
ws_set_cursorPixmap.....	14	
ws_get_cursor_color	14	
ws_set_cursor_color.....	15	
ws_enable_inputdevice	15	
ws_get_events	15	
3	The libPK library	16
3.1	Data Structures.....	16
Pk_board	16	
Pk_colormap.....	16	
Pk_drawable.....	16	
Pk_font	17	
Pk_glyph_info	17	
Pk_pixelinfo	17	
3.2	Pk_board Methods	19
pk_create_board	19	

pk_get_boardinfo	20
pk_set_framebuffer	20
pk_get_videosize.....	21
pk_get_drawable	21
pk_get_errorcode.....	21
pk_set_errorcode	21
pk_create_drawable.....	22
pk_get_cursordrawable	22
pk_set_cursorhotspot.....	22
pk_position_cursor.....	22
pk_create_truetype_font	22
pk_create_ti_font.....	22
pk_create_ti_font_mem.....	23
pk_video_enable.....	23
pk_swap_buffers	23
pk_pan_absolute.....	23
pk_pan_relative	23
pk_get_pixelinfo.....	24
pk_read_flash.....	24
pk_write_flash.....	24
pk_read_seeprom.....	24
pk_write_seeprom.....	24
3.3 Pk_colormap Methods	25
pk_lookup_color.....	25
pk_allocate_color.....	25
pk_allocate_closest_color	25
pk_allocate_colorcell.....	25
pk_allocate_colorcells.....	25
pk_release_color.....	26
pk_release_colors.....	26
pk_allocate_colorplanes.....	26
pk_release_colorplanes	26
pk_set_color.....	26
pk_set_colors.....	26
pk_set_colors_random.....	27
pk_get_color.....	27
pk_get_colors.....	27
pk_get_colors_random.....	27
3.4 Pk_drawable Methods	27
pk_get_board.....	27
pk_front_drawable	27
pk_back_drawable.....	28
pk_primary_drawable.....	28
pk_overlay_drawable	28
pk_create_front_drawable.....	28
pk_create_back_drawable	28

pk_create_primary_drawable.....	28
pk_create_overlay_drawable.....	28
pk_destroy_drawable	29
pk_get_size.....	29
pk_get_cliprect.....	29
pk_get_pixformat.....	29
pk_get_depth.....	29
pk_get_memspace	29
pk_set_cliprect	29
pk_make_color.....	30
pk_extract_color.....	30
pk_read_pixel.....	30
pk_draw_pixel.....	30
pk_draw_line.....	30
pk_draw_triangle.....	30
pk_draw_rectangle	30
pk_draw_polygon.....	31
pk_draw_polyline.....	31
pk_draw_linesegs.....	31
pk_draw_circle.....	31
pk_draw_ellipse	31
pk_draw_arc.....	31
pk_draw_piearc	32
pk_fill_triangle.....	32
pk_fill_rectangle.....	32
pk_fill_polygon.....	32
pk_fill_circle	32
pk_fill_ellipse.....	33
pk_fill_piearc	33
pk_create_colormap.....	33
pk_get_colormap.....	33
pk_destroy_colormap.....	33
pk_draw_char.....	33
pk_draw_text	33
pk_draw_text16.....	34
pk_draw_text32	34
pk_draw_string.....	34
pk_draw_string16.....	34
pk_draw_string32.....	34
pk_bitblt	35
pk_block_copy.....	35
pk_shaded_line.....	35
pk_shaded_triangle.....	36
pk_read_span.....	36
pk_write_span.....	36
pk_readPixmap.....	36

pk_write_pixmap.....	36
pk_flush.....	37
3.5 Pk_font Methods.....	37
pk_destroy_font.....	37
pk_get_ascent.....	37
pk_get_descent.....	37
pk_get_spacing.....	37
pk_glyph_info	37
4 The libVFX library.....	39
4.1 Configuration and test.....	39
VxWorks, MVME2604.....	39
Running demos on VxWorks PowerPC systems	41
4.2 Pixel format	42
4.3 Memory spaces	43
4.4 Board info	43
4.5 VFX-M specific functions	44
vfx_load_timing	44
vfx_show_timing.....	44
vfx_write_palette.....	44
5 The libVCQM library	46
5.1 Configuration and test.....	46
VxWorks, MVME2604.....	46
Running demos on VxWorks PowerPC systems	48
5.2 Pixel format	49
5.3 Memory spaces	50
5.4 Board info	50
5.5 VCQ-M specific functions.....	50
vcqm_access_registers.....	50
vcqm_configure.....	51
vcqm_load_timing.....	51
vcqm_show_timing.....	51
vcqm_write_palette	52
vcqm_write_cursor_color.....	52
6 The libPKMesa library	53
6.1 Data structures	53
PKMesaVisual.....	53
PKMesaBuffer.....	53
PKMesaContext	53
6.2 Functions.....	53
PKMesaCreateVisual	53
PKMesaDestroyVisual.....	53
PKMesaCreateBuffer	54
PKMesaDestroyBuffer.....	54
PKMesaCreateContext.....	54
PKMesaDestroyContext.....	54
PKMesaMakeCurrent	54

PKMesaSwapBuffers	54
7 The libGL library	54
Base OpenGL functions	55
OpenGL extensions	57
Mesa specific OpenGL extensions	58
SGI OpenGL extensions	58
8 The libGLU library	58
9 The libGLUT library	58
10 The libTrueType library.....	58

1 Introduction

1.1 Overview of the Graphics Library Package (GLP)

The Graphics Library Package (GLP) is designed to allow portability of code between different Peritek PCI and PMC graphics boards without sacrificing the ability to program the graphics hardware directly for applications that require it.

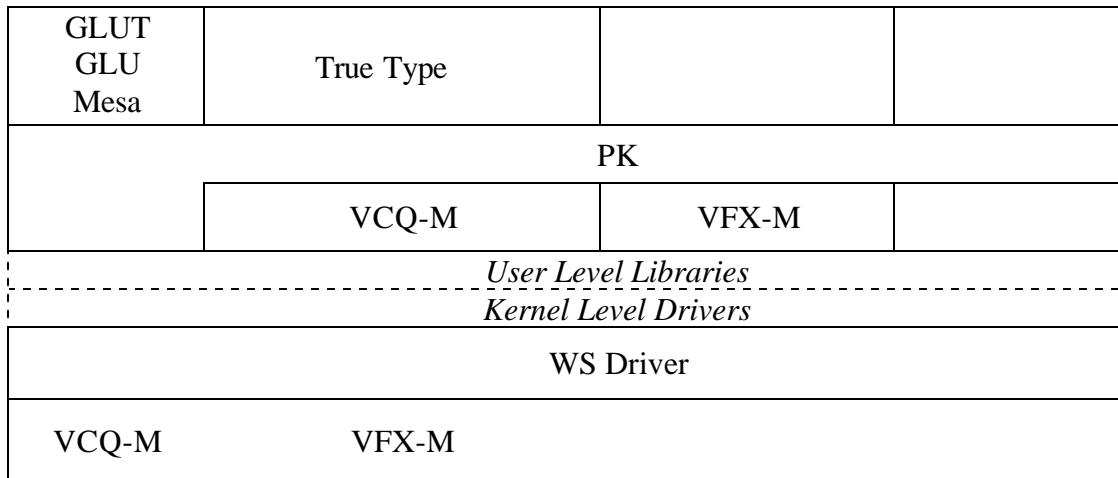


Figure 1-1 GLP Software Structure

The core of the package is the **libPK** (Peritek) library. This library provides an API (Applications Programming Interface) for generic rendering, but with the capability of using optimized methods for each Peritek board type. To assist with system initialization, the **libWS** (workstation) library provides a system-independent interface for accessing graphics and I/O device drivers. Support for the OpenGL API is provided with **libPKMesa**, based on the publicly available Mesa 3-D graphics package. Support for GLU and GLUT facilities will be provided with **libGLU** and **libGLUT**, and True Type font support will be provided with **libTT**. Figure 1-1 illustrates the GLP's software structure. Higher level libraries are implemented using those shown below them in the figure. Other higher level libraries not shown may be implemented in the future. The **libWS**, **libGLU**, **libGLUT** and **libTT** libraries are not yet available.

Library Name	Purpose	Function prefix
libWS	Access to the ws driver	ws_
libPK	Generic rendering	pk_
libVFX	VFX-M specific rendering	vfx_
libVCQM	VCQ-M specific rendering	vcqm_
libPKMesa	Peritek Mesa support for OpenGL API	PKMesa
libGLU	OpenGL utilities	glu_
libGLUT	OpenGL tools	glut_
libTrueType	True Type font rendering	TT_

Figure 1-2 Naming Conventions for Libraries

The **GLP** is composed of a number of modules and libraries. Each library and driver module has its own naming convention (see Figure 1–2 and Figure 1–3). The underscore after the prefix in a file name indicates that it is part of a Peritek library. The Mesa library uses a different convention. Test programs and demos that use a library are often named with the prefix but without the underscore.

Device Driver	Purpose	Function prefix
ws	Workstation driver/manager	wsdrv_
vfx	VFX-M device driver	vfx_
vcq	VCQ-M device driver	vcqm_

Figure 1–3 Naming Convention for Drivers

1.2 Installation

The Peritek **GLP** is supplied on CDROM and includes precompiled library archives as well as source code. The software can be found on the CDROM both as a **tar** file (usually the fastest and easiest format for installation) and unpacked, for reference. To install on a system where **tar** is available, do the following:

- 1) Copy the file **glp.tar** from the CDROM to the directory where you wish to keep the Peritek software tree.
- 2) Extract the file hierarchy from the tar file: `tar xf glp.tar`

If you do not have **tar** on your system, copy the file hierarchy directly from the CDROM.

The locations of **GLP** source code, binaries and scripts in the file hierarchy are shown in the following table. Path names are shown relative to the directory where you installed the package.

Directory	Description of contents
fonts/	Font files
lynxos/	The pciacc.c driver accessing the PCI bus on Motorola
mesa/	The Mesa-3.0 distribution
obj/	Object files
obj/vxw-mv2604/	VxWorks MVME2604 specific object files
obj/vxw-ppc/	VxWorks PowerPC specific object files
obj/vxw-ppc/mesa-src/	Object files for Mesa
src/	Source files
src/glpl/	GLP source files (libPK , libVFX , libVCQM)
	Makefiles
src/glpl/GL/	Mesa libGL header files
src/glpl/mesa-src/	Mesa libGL source files
vxworks/	Tornado shell scripts specific to VxWorks

1.3 Getting started

Once the GLP software has been installed in your system, as described in the previous section, check the **Configuration and Testing** sections for your board type (see section 5.1 for the VCQ-M and section 4.1 for the VFX-M.) You may need to install a driver, reconfigure your kernel for more memory space, or run a configuration routine to set up devices on the PCI bus, before you can run the Peritek-supplied demos or graphics applications of your own. These sections will also describe how to determine the bus and device IDs of your graphics boards, which will vary according to the configuration of your processor board. These bus and device IDs must be given as parameters to **pk_create_board** in order to map the graphics board.

After your system has been configured, you should run the Peritek-supplied pre-compiled demo and test programs shown in Figure 1–4. Scripts are provided to run these programs on your operating system and to serve as examples of the arguments for each demo program. See the **Configuration and Test** section for your board (section 5.1 for the VCQ-M and section 4.1 for the VFX-M.) for more information about the scripts to run for your board type and operating system.

Demo program	Description	Source files
pkflag	Displays a flag pattern with labels	pkflag.c
pktestpat	Displays a test pattern created with libPK on the specified board.	pktestpat.c
pkdemo	Displays a demo that draws shaded teapots, using libPKMesa on the specified board	pkdemo.c
pkmulti	A single program that controls multiple VCQ-M channels using libPK .	pkmulti.c pkflag.c pkhello.c

Figure 1–4 Precompiled demonstration and test programs

Once you are able to run the demos correctly, begin developing your own application by recompiling the demos in directory **src/glp/**. A sample Makefile is provided for your operating system and processor board type, indicated by the extension, e.g. **Makefile.vxw-ppc-mv2604..** You will need to change the Makefile, setting environment variables and adjusting paths to reflect your system.

After successfully recompiling and running the demos, you are ready to start developing your own application using **libPK** or **libPKMesa**. For **libPK**, begin by reading *carefully* the description of the library's data structures in Section 3.1. The **Pk_board** data structure represents the display device, which may in fact be only part of a physical board. The VCQ-M, for example, has four display controllers, and thus four possible **Pk_board** structures for each physical board. The **pk_create_board** method

associates a **Pk_board** structure with a particular device, and **pk_set_framebuffer** initializes the display to a particular resolution, sets the pixel format and also initializes a structure of type **Pk_drawable**.

The concept of drawable represented in the structure **Pk_drawable** includes both a region of video memory and a set of methods that will act on the drawable whenever the **libPK** interface for a method is called on that drawable. Derived drawables may operate on the same region of memory as a base drawable, but with a different set of methods. Planned development for the library includes using derived drawables to implement double-buffering, pen and pattern drawing, and viewports. Drawables may also refer to a part of video memory that is not displayed. Drawing methods may be used on this memory in the same way as for the visible frame buffer, and the resulting picture moved to the display drawable using bitblt or block copy.

The main display drawable created by **pk_set_framebuffer** is initialized to refer to all of the displayable primary video memory (the actual display, plus any memory that may be displayed using pan and scroll), and to the set of methods that allows access to that board type. In order to draw something on the primary planes of the display, use **pk_get_drawable** with the board data structure as argument to get a pointer to this main display drawable, and then use any desired drawing method with that drawable as an argument. Calling **pk_overlay_drawable** with the drawable pointer returned by **pk_get_drawable** as argument returns a pointer to a drawable that can be used to operate on displayable overlay video memory.

For **libPK**, scan the lists of board, colormap, drawable and font methods for the operations you want to perform, and use the source code for the demos as examples to get started. For **libPKMesa**, see the references to documentation on the Mesa 3-D library and the OpenGL API in sections 6 and 7.

1.4 Using this manual

Functions available for you to use in developing your graphics application are described in this manual, grouped by library. Within the library description sections, related functions are grouped together. To look up a function by name, use the Index at the back of the manual. You will also find certain topics listed in the Index, such as “colormap” or “pixel format.”

Different fonts are used as follows:

Bold face type (like this) indicates a special name within running text, such as a function name, a structure name, a file name, or a library name.

Plain fixed width text (like this) indicates a code fragment, such as an exact function declaration.

Italic type (like this) is used for referencing the names of function arguments within running text describing the function.

You may also wish to consult the hardware manual for your board type.

2 The libWS library

The WS model is primarily a mechanism for dividing a set of UI input and output devices into a set of workstations. Each workstation is composed of a set of input and output devices. There may be one or more workstations per system. The WS driver control device is used to configure the WS driver and allocate devices to workstations. If the OS console is a serial port then at OS boot time no workstations are configured. Otherwise, a single workstation is configured to include the console display and keyboard. At OS boot time each physical device that is capable of being managed by the WS driver must register its capabilities with the WS driver.

The libWS library is not yet implemented, and the following documentation is not yet complete. Until it is completed, applications may need to use configuration routines for each specific board type as well as **pk_create_board** to access the devices.

2.1 Data Structures

Ws_device

Ws_workstation

Ws_region

Ws_video_timing

Ws_event

2.2 Functions

ws_registered_devices

```
int ws_registered_devices(Ws_device *devices, int maxdevs);
```

Query a list of devices that have been registered with the ws driver.

Devices may be of the following classes: displays, keyboards, pointers, miscellaneous input, audio in, audio out, and video input. The current set of supported devices is as follows:

Manufacture	Model	Device Class	Device Type
Peritek	VFX-M	display	VFX
Peritek	VFX-M	keyboard	PS/2 keyboard
Peritek	VFX-M	pointer	PS/2 mouse
Peritek	VCQ-M	display	VCQ
Peritek	VCQ-M	keyboard	USB keyboard
Peritek	VCQ-M	pointer	USB pointer

For each device that is currently registered, **ws_registered_devices** returns the device's unique name, a text description of the device, the number of the device class, and the numeric device type. See **ws.h** for the current set of names for the numeric device classes and types.

The *maxdevs* parameter specifies the length of the *devices* array. Information on up to *maxdevs* devices can be returned. The return value is *-1* if an error occurred, or the number of registered devices that information was returned about.

ws_configured_workstations

```
int ws_configured_workstations(Ws_workstation *stations, int maxws);
```

Query the list of workstation devices that have been configured. A configured workstation is one that has one or more devices attached to it. The name of each configured workstation is returned.

The *maxws* parameter specifies the length of the *stations* array. Information on up to *maxws* workstations can be returned. The return value is *-1* if an error occurred, or the number of configured workstations that information was returned about.

ws_configured_devices

```
int ws_configured_devices(int ws, Ws_device *devices, int maxdevs);
```

Query the list of devices that have been attached to a particular workstation. The *ws* parameter is a workstation id as returned by the **ws_open_workstation** function.

The *maxdevs* parameter specifies the length the *devices* array. Information on up to *maxdevs* devices can be returned. The return value is *-1* if an error occurred, or the number of registered devices that information was returned about.

ws_open_workstation

```
int ws_open_workstation(char *name);
```

Returns the workstation id of the opened workstation or *-1* on an error.

ws_close_workstation

```
int ws_close_workstation(int wd);
```

Close the workstation device associated with the workstation id *wd*.

ws_get_video_timing

```
int ws_get_video_timing(int wd, int dev, Ws_video_timing *timing);  
  
ws_set_video_timing  
int ws_set_video_timing(int wd, int dev, Ws_video_timing *timing);  
  
ws_get_regioninfo  
int ws_get_regioninfo(int wd, int dev, Ws_region *regions, int  
                      maxregions);  
  
ws_map_region  
int ws_map_region(int wd, int dev, int region, void **ptr);  
  
ws_get_colormap_random  
int ws_get_colormap_random(int wd, int dev, int *index, int *red, int  
                           *green, int *blue);  
  
ws_set_colormap_random  
int ws_set_colormap_random(int wd, int dev, int *index, int *red, int  
                           *green, int *blue);  
  
ws_get_colormap  
int ws_get_colormap(int wd, int dev, int *index, int *red, int *green,  
                     int *blue);  
  
ws_set_colormap  
int ws_set_colormap(int wd, int dev, int index, int count, int *red,  
                     int *green, int *blue);  
  
ws_associate_cursor  
int ws_associate_cursor(int wd, int pointingdev, int cursornum);  
  
ws_get_cursorinfo  
int ws_get_cursorinfo(int wd, int cursornum, Ws_cursor *cursor);  
  
ws_set_cursor_pixmap  
int ws_set_cursor_pixmap(int wd, int cursornum, int width, int height,  
                        int hotx, int hoty, unsigned char *pixmap);  
  
ws_get_cursor_color  
int ws_get_cursor_color(int wd, int cursornum, int colornum, int *red,  
                        int *green, int *blue);
```

ws_set_cursor_color

```
int ws_set_cursor_color(int wd, int cursornum, int colordnum, int red,  
int green, int blue);
```

ws_enable_inputdevice

```
int ws_enable_inputdevice(int wd, int dev, int enable);
```

ws_get_events

```
int ws_get_events(int wd, Ws_event *events, int maxevents);
```

3 The libPK library

The **libPK** library is a graphics card independent API for doing 2D and some 3D graphics. The API is close to the hardware. For the most part the API is implemented by macros that indirectly call the correct board specific function. The **libPK** library also provides generic implementations of various graphics functions. It is possible to use the API to draw on just about anything. Specializations and optimizations are done for the particular capabilities of the VFX-M and VCQ-M, based on the hardware dependent libraries.

3.1 Data Structures

Pk_board

This structure has no members that are accessible to the user. It is used to represent a specific display device. For the VFX-M it contains information necessary to access the Number Nine i128 series II graphics controller. For the VCQ-M it contains information necessary to access one of the four Cirrus Logic GD7555 graphics controllers. The **pk_create_board** function returns a pointer to a **Pk_board** object. After calling **pk_destroy_board** no further use should be made of the object. The function **pk_set_framebuffer** is used to initialize a board structure.

Pk_colormap

This structure has no members that are accessible to the user. It is used to represent a color palette. A **Pk_colormap** can be associated with every **Pk_drawable**. The **Pk_colormap** contains a mapping from pixel values to the red, green, and blue components of each color. A colormap may be used either for indexed color lookup or for gamma correction.

Pk_drawable

This structure is used to represent either a chunk of memory that is capable of storing an array of pixels and a set of methods for rendering pixels in a particular fashion, or else a set of methods for rendering on another base drawable. There are a number of drawing methods that can be used on a drawable to generate graphics. For different kinds of drawables, the **libPK** method interfaces described in section 3.4 will invoke the appropriate method for that drawable and board type. To make pictures appear on the screen, use **pk_get_drawable**, with an initialized **Pk_board** pointer as its argument, to get the primary display drawable. Then operate on that drawable with colormap and drawing methods, and see the result directly on the screen. Sophisticated applications create off-screen drawables using the function **pk_create_drawable**, operate on the off-screen drawable with drawing methods, and then transfer pixels to the screen using **pk_block_copy** or **pk_bitblt**. On boards that support overlays, the function **pk_overlay_drawable** can be applied to the

primary display drawable to get a drawable that can be used to display graphics in the overlay.

Pk_font

This structure is used to represent a font that can be used for drawing text. The structure contains no user accessible members. Currently only TI fonts can be represented. Support for TrueType fonts will be added in the future.

Pk_glyph_info

This structure is used to communicate information about the geometry of a glyph in a font. Its user accessible members are:

`unsigned int code`

This member is the character code of the glyph in the font. For example, the character code for the ASCII letter ‘A’ is 65. The type of the field is `unsigned int` so that 16-bit and 32-bit character codes can be supported.

`int width`

This member is the width of the glyph bitmap. The width is the number of pixels between and including the left most drawn pixel and the right most drawn pixel of the glyph.

`int height`

This member is the height of the glyph bitmap. The height is the number of pixels between and including the top most drawn pixel and the bottom most drawn pixel of the glyph.

`int refx`

This member is the signed offset from the left most drawn pixel of the glyph to the glyphs reference point. This value will be positive if the reference point is to the right of the left most drawn pixel. It will be negative if the reference point is to the left of the glyph image.

`int refy`

This member is the signed offset from the top most drawn pixel of the glyph to the glyphs reference point. This value will be positive if the reference point is below the top most drawn pixel. It will be negative if the reference point is above the glyph image.

`int space`

This member is the number of pixels from the reference point of the current character to the reference point of the next character without kerning.

Pk_pixelinfo

This structure is used to represent information that describes the structure of a pixel for a particular pixel format.

```
int format
```

This member is the id of the specific pixel format. This id is specific to the particular graphics controller that the pixel info was derived for.

```
int psize
```

This member is the width in bits of a pixel. Generally it is either 1, 2, 4, 8, 16, 24, or 32. It represents the number of bits of memory that the pixel occupies which may be greater than the number of bits that contain useful data.

```
unsigned int mask
```

This member is a mask that can be used to extract just the pixel bits from a word that has been adjusted so that pixel bit zero is in bit zero of the word. It is almost always equivalent to `((1 << psize) - 1)`.

```
int pindexed
```

This flag is true if the primary planes of the pixel data are indexed through a colormap.

```
int pwidth
```

This member is the width of the primary planes portion of the pixel data. For pixel formats that don't include overlay data this is generally the same as the pixel size.

```
int pshift
```

This is the number of bits of non-primary pixel data that are in the least significant bits of the pixel data.

```
unsigned int pmask
```

This is a mask that can be used to extract just the primary pixel data out of a pixel.

```
unsigned int pdmask
```

This is a mask that can be used to extract just the primary pixel bits from a word that possibly contains multiple pixels.

```
int rsize
```

This is the width of the red field for non-indexed pixels or the index field for indexed pixels.

```
int rshift
```

This is the number of bits from the least significant bit of the pixel to the least significant bit of the red field.

```
unsigned int rmask
```

This is a mask that can be used to extract just the red field from the pixel data.

```
int gsize
```

This is the width of the green field of non-indexed pixels. For indexed pixels this field must be zero.

```
int gshift
```

This is the number of bits from the least significant bit of the pixel to the least significant bit of the green field.

`unsigned int gmask`

This is a mask that can be used to extract just the green field from the pixel data.

`int bsize`

This is the width of the green field of non-indexed pixels. For indexed pixels this field must be zero.

`int bshift`

This is the number of bits from the least significant bit of the pixel to the least significant bit of the blue field.

`unsigned int bmask`

This is a mask that can be used to extract just the blue field from the pixel data.

`int oindexed`

This field is true if the overlay portion of the pixel data is indexed.

`int owidth`

This field is the width of the overlay data. It must be zero if the pixel format does not include overlay data.

`int oshift`

This is the number of bits of non-overlay pixel data that are in the least significant bits of the pixel data.

`unsigned int omask`

`unsigned int odmask`

`int orsize`

`int orshift`

`unsigned int ormask`

`int ogsize`

`int ogshift`

`unsigned int ogmask`

`int obsize`

`int obshift`

`unsigned int obmask`

These fields have the same meaning as the respective field for the primary pixel data, except these are used to describe the overlay pixel data.

3.2 Pk_board Methods

pk_create_board

```
int pk_create_board(char *devname, int bus, int dev, int func, int
    *errcode);
```

This function takes as parameters the information necessary to specify a particular Peritek PCI or PMC graphics board in a system and returns a pointer to an object that

can be used to access that board. The information necessary to specify a particular board is the device file name, if any, and the device's bus number, device number and function number. The bus, device, and function numbers are physical properties of the PCI bus. The bus and device numbers are system specific. The Peritek VFX-M and VCQ-M boards are single function devices, according to the terminology of the PCI standard, and have a function number of 0. Returns 0 on error.

pk_get_boardinfo

```
int pk_get_boardinfo(Pk_board *board, int info, unsigned int *value);
```

Query specific properties of the graphics board. The following values for *info* are supported for all boards.

`PK_BOARD_INFO_BOARDTYPE`

Get type of graphics board referenced by this Pk_board object. The result returned in the location pointed to by *value* will be one of the following constants.

`PK_BOARD_TYPE_HOSTMEM`
`PK_BOARD_TYPE_VFX`
`PK_BOARD_TYPE_VCQM`

Returns 0 on error.

pk_set_framebuffer

```
int pk_set_framebuffer(Pk_board *board, int width, int height, int freq, int vwidth, int vheight, int pixformat, int doublebuf);
```

This method sets the frame buffer geometry, allocates space out of the video memory array for the frame buffer, allocates the optional back buffer, programs the video timing registers, PLL's and RAMDAC control registers.

Width and *height* are the size of the visible area that is to be displayed on the monitor. The granularity of control over the width of the visible display area varies by graphics board type, configuration, and pixel format. The current worst case resolution is 16-pixels in 8bpp mode on the VFX-M. If the value of the *width* parameter is not a multiple of the granularity then the *width* will be rounded up to the next value that is a multiple of the granularity.

Freq is the display frame rate in 1/1000 Hertz increments. For a 60Hz refresh rate the *freq* parameter should be 60,000.

Vwidth and *vheight* are the width and height of the virtual frame buffer. They control the amount of video memory allocated for the frame buffer. They must always be greater than or equal to the size of the display area. If they are greater than the size of

the display area then the *pan* methods may be used to pan the displayed region within the virtual frame buffer.

Pixformat is a numeric value that controls the format of pixels in the frame buffer. See the section for your particular graphics card for a list of pixel formats that are supported.

Doublebuf controls whether an optional back buffer is allocated out of video memory. If this parameter is non-zero then the back buffer will be allocated, otherwise it will not be. The displayed buffer can be changed with the **pk_swap_buffers** method.

Returns 0 on error.

pk_get_videosize

```
int pk_get_videosize(Pk_board *board, int *width, int *height);
```

Returns the width and height of displayed video memory in pixels. The values are returned in the locations pointed to by the *width* and *height* parameters. Returns non-zero if a valid width and height are returned. Otherwise, zero is returned. Zero will be returned if no call has been made to *pk_set_framebuffer*.

pk_get_drawable

```
Pk_drawable *pk_get_drawable(Pk_board *board);
```

Returns a pointer to the special drawable that represents the displayable video memory of a board. This drawable always represents the primary planes. The drawable for the overlay planes can be found by calling **pk_overlay_drawable** on the returned drawable. If display memory has not yet been configured with a call to **pk_set_framebuffer**, a null pointer is returned.

pk_get_errorcode

```
int pk_get_errorcode(Pk_board *board);
```

Most functions do not return specific error status, either no error indication is returned at all, or a zero or negative one (-1) is returned to indicate that an error occurred. The **pk_get_errorcode** function allows for a more specific error indication to be returned. Depending upon the error code that is returned additional error status may be available.

pk_set_errorcode

```
void pk_set_errorcode(Pk_board *board, int code);
```

This function sets the current error code. The error code can be queried with the `pk_get_errorcode` function.

pk_create_drawable

```
Pk_drawable *pk_create_drawable(Pk_board *board, int width, int height,  
                                int pixformat, int memspace);
```

This function is used to create an off screen drawable. The *width* and *height* parameters control the size of the off screen drawable. The *pixformat* parameter is specific to the graphics board in use; the section for the specific graphics board should be consulted. For boards that have multiple memories that can be rendered in (i. e. the VFX-M has both 8MB of VRAM and 64MB of DRAM) the *memspace* parameter specifies the memory out of which to allocate the off screen drawable. If there is insufficient space to allocate the off screen drawable, then a null pointer will be returned. The drawable should released when it is no longer needed by calling `pk_destroy_drawable`.

pk_get_cursordrawable

```
int pk_get_cursordrawable(Pk_board *board);
```

Returns a pointer to the cursor drawable. Returns 0 on error.

pk_set_cursorhotspot

```
int pk_set_cursorhotspot(Pk_board *board, int x, int y);
```

Sets the location of the cursor hot spot. This is the point in the cursor that will be positioned by the `pk_position_cursor` method. The hot spot is relative to the upper left hand corner of the cursor image. Returns 0 on error.

pk_position_cursor

```
int pk_position_cursor(Pk_board *board, int x, int y);
```

Positions the cursor hot spot to the display coordinates (*x*, *y*). Returns 0 on error.

pk_create_truetype_font

```
Pk_font *pk_create_truetype_font(Pk_board *board, char *filename);
```

This function is used to create a font object. The font information will come from the TrueType font file named *filename*. The returned font object will have methods that are specific to both the graphics board and TrueType fonts.

pk_create_tifont

```
Pk_font *pk_create_ti_font(Pk_board *board, char *filename);
```

Return a font object for the TI font data located in the file named *filename*.

pk_create_ti_font_mem

```
Pk_font *pk_create_ti_font_mem(Pk_board *board, unsigned char *font,
                                int fontlen);
```

Returns a font object for the TI font data located in memory at location *font*. The font data is *fontlen* bytes long.

pk_video_enable

```
void pk_video_enable(Pk_board *board, int onoff);
```

The *onoff* parameter controls whether video output is enabled (on) or disabled (off). When video is disabled, horizontal and vertical syncs may or may not be disabled depending upon the model of the graphics board.

pk_swap_buffers

```
void pk_swap_buffers(Pk_board *board, int wait);
```

The displayed buffer is swapped between the front buffer and the back buffer. If the front buffer is currently displayed then the back buffer will be displayed upon return. If the back buffer is currently displayed then the front buffer will be displayed upon return. The *wait* parameter controls whether it is necessary to wait for vertical sync prior to swapping the buffers.

pk_pan_absolute

```
void pk_pan_absolute(Pk_board *board, int x, int y);
```

If the width or height of the displayed drawable is larger than the width or height of the display, then the display may be panned within the area of the display drawable.

The *x* parameter is the pan offset in pixels along the x-axis. Adjusting the pan position from 16 to 32 causes the display to appear to move to the left. The x-axis pan increment may be granular in which case the display position can only be adjusted in multiples of some number of pixels, this is usually a power of two.

The *y* parameter is the pan offset in pixels along the y-axis. Adjusting the pan position from 1 to 2 causes the display to appear to move up.

pk_pan_relative

```
void pk_pan_relative(Pk_board *board, int deltax, int deltay);
```

This function is similar *pk_pan_absolute*, except that they x and y-axis offsets are relative to the current pan position.

pk_get_pixelinfo

```
Pk_pixelinfo *pk_get_pixelinfo(Pk_board *board, int pixformat);
```

A pixelinfo structure is returned for the pixel format specified by *pixformat*. The pixel information returned is graphics board specific. The layout of the pixelinfo structures was described earlier. A null pointer is returned if the pixel format is not supported.

pk_read_flash

```
int pk_read_flash(Pk_board *board, int offset, unsigned char *buf, int len);
```

The contents of the boards BIOS flash memory beginning at *offset* are copied into the buffer *buf*. Up to *len* bytes are copied. The number of bytes actually copied is returned.

pk_write_flash

```
int pk_write_flash(Pk_board *board, int offset, unsigned char *buf, int len);
```

The contents of the boards BIOS flash memory beginning at *offset* are programmed from the contents of the buffer *buf*. Up to *len* bytes are programmed. The number of bytes actually programmed is returned.

Board Type	BIOS flash memory size
VFX-M	128Kb
VCQ-M	128Kb

pk_read_seeprom

```
int pk_read_seeprom(Pk_board *board, int offset, unsigned char *buf, int len);
```

The contents of the boards serial EEPROM memory beginning at *offset* are copied into the buffer *buf*. Up to *len* bytes are copied. The number of bytes actually copied is returned.

pk_write_seeprom

```
int pk_write_seeprom(Pk_board *board, int offset, unsigned char *buf, int len);
```

The contents of the boards serial EEPROM memory beginning at *offset* are programmed from the contents of the buffer *buf*. Up to *len* bytes are programmed. The number of bytes actually programmed is returned.

Board Type	Serial EEPROM memory size
VFX-M	256 bytes
VCQ-M	256 bytes

3.3 Pk_colormap Methods

pk_lookup_color

```
unsigned int pk_lookup_color(Pk_colormap *cmap, int r, int g, int b);
```

Returns the index of the color cell that contains the color closest to the requested color. The *r*, *g*, and *b* fields specify the pixel colors and must contain value between zero and 65535. Zero represents 0% of the color, 65535 represents 100% of the color.

pk_allocate_color

```
unsigned int pk_allocate_color(Pk_colormap *cmap, int r, int g, int b);
```

Returns the index of a color cell that contains the specified red, green, and blue values. The color in the color cell can not be changed while the colorcell is in use.

pk_allocate_closest_color

```
unsigned int pk_allocate_closest_color(Pk_colormap *cmap, int r, int g, int b);
```

Returns the index of a color cell that contains either the specified red, green, and blue values or red, green and blue values that are “close” to the specified color. The color in the color cell can not be changed while the colorcell is in use.

pk_allocate_colorcell

```
unsigned int pk_allocate_colorcell(Pk_colormap *cmap);
```

Returns the index of an uninitialized color cell. **pk_set_color** can be used to set the color of the color cell. The returned color cell will not be reused by **pk_allocate_color** or **pk_allocate_closest_color**.

pk_allocate_colorcells

```
unsigned int pk_allocate_colorcells(Pk_colormap *cmap, int numcells);
```

Same as **pk_allocate_colorcell** except that multiple contiguous colorcells are allocated in a single call.

pk_release_color

```
int pk_release_color(Pk_colormap *cmap, unsigned int index);
```

Releases a color cell allocated with **pk_allocate_color**, **pk_allocate_closest_color**, or **pk_allocate_colorcell**. Returns 0 on error.

pk_release_colors

```
int pk_release_colors(Pk_colormap *cmap, unsigned int index, int numcolors);
```

Releases a contiguous set of colorcells that were allocated with a call to **pk_allocate_colorcells**. Returns 0 on error.

pk_allocate_colorplanes

```
unsigned int pk_allocate_colorplanes(Pk_colormap *cmap, int numred, int numgreen, int numblue);
```

Allocates one or more bits of the pixel data.

pk_release_colorplanes

```
int pk_release_colorplanes(Pk_colormap *cmap, unsigned int planes);
```

Releases one or more bits of pixel data that were allocated with **pk_allocate_colorplanes**. Returns 0 on error.

pk_set_color

```
int pk_set_color(Pk_colormap *cmap, unsigned int index, int r, int g, int b);
```

Changes the color stored in a color cell allocated with **pk_allocate_colorcell**, **pk_allocate_colorcells**, or **pk_allocate_colorplanes**. Returns 0 on error.

pk_set_colors

```
int pk_set_colors(Pk_colormap *cmap, unsigned int index, int num, short *r, short *g, short *b);
```

Changes the colors stored in one or more contiguous color cells allocated with **pk_allocate_colorcells**. The values in location 0 to *num - 1* in the arrays *r*, *g* and *b* will be mapped to pixel values *index* to *index + num - 1*. Returns 0 on error.

pk_set_colors_random

```
int pk_set_colors_random(Pk_colormap *cmap, int num, unsigned int
    *index, short *r, short *g, short *b);
```

Changes the color stored in one or more color cells allocated with **pk_allocate_colorcell**, **pk_allocate_colorcells**, or **pk_allocate_colorplanes**. The *num* pixel values returned in the array *index* are mapped to the values from 0 to *num* - 1 in the arrays *r*, *g* and *b*. Returns 0 on error.

pk_get_color

```
int pk_get_color(Pk_colormap *cmap, unsigned int index, short *r, short
    *g, short *b);
```

Returns the red, green and blue values stored in a color cell in *r*, *g* and *b*. Returns 0 on error.

pk_get_colors

```
int pk_get_colors(Pk_colormap *cmap, unsigned int index, int num, short
    *r, short *g, short *b);
```

Returns the colors stored in one or more contiguous color cells. Returns 0 on error.

pk_get_colors_random

```
int pk_get_colors_random(Pk_colormap *cmap, int num, unsigned int
    *index, short *r, short *g, short *b);
```

Returns the colors stored in one or more color cells. The *index* array contains the indices of the color cells that should be used to retrieve the *r*, *g* and *b* values. Returns 0 on error.

3.4 Pk_drawable Methods

pk_get_board

```
Pk_board *pk_get_board(Pk_drawable *drawable);
```

Returns a pointer to the board associated with a drawable.

pk_front_drawable

```
Pk_drawable *pk_front_drawable(Pk_drawable *drawable);
```

Returns a pointer to the drawable itself, if it is a front drawable, or to the associated front drawable. Used when double buffering.

pk_back_drawable

```
Pk_drawable *pk_back Drawable(Pk_drawable *drawable);
```

Returns a pointer to the drawable itself, if it is a back drawable, or to the associated back drawable. Used when double buffering.

pk_primary_drawable

```
Pk_drawable *pk_primary Drawable(Pk_drawable *drawable);
```

Returns a pointer to the drawable itself, if it is a primary drawable, or to the associated primary drawable if the input drawable is an overlay drawable.

pk_overlay_drawable

```
Pk_drawable *pk_overlay Drawable(Pk_drawable *drawable);
```

Returns a pointer to the drawable itself, if it is an overlay drawable, or to the overlay display drawable associated with the input primary drawable, if any.

pk_create_front_drawable

```
Pk_drawable *pk_create_front Drawable(Pk_drawable *drawable);
```

Create a front drawable from the input drawable. Returns NULL on failure.

pk_create_back_drawable

```
Pk_drawable *pk_create_back Drawable(Pk_drawable *drawable);
```

Creates a back drawable from the input drawable. Returns NULL on failure.

pk_create_primary_drawable

```
Pk_drawable *pk_create_primary Drawable(Pk_drawable *drawable);
```

Create a primary display drawable from the input drawable. Returns NULL on failure.

pk_create_overlay_drawable

```
Pk_drawable *pk_create_overlay Drawable(Pk_drawable *drawable);
```

Create an overlay display drawable from the input drawable. Returns NULL on failure.

pk_destroy_drawable

```
void pk_destroy_drawable(Pk_drawable *drawable);
```

Destroys the input drawable; the pointer to the drawable should not be referenced after this call.

pk_get_size

```
int pk_get_size(Pk_drawable *drawable, int *width, int *height);
```

Sets *width* and *height* to point to the width and height of the drawable in pixels. Returns 0 on error.

pk_get_cliprect

```
int pk_get_cliprect(Pk_drawable *drawable, int *left, int *top, int  
*right, int *bottom);
```

Sets *left*, *top*, *right* and *bottom* to point to the bounds of the clipping rectangle associated with the drawable. Returns 0 on error.

pk_get_pixformat

```
int pk_get_pixformat(Pk_drawable *drawable);
```

Returns the pixel format in use on the drawable. Pixel formats available for each board are described in the section for that board.

pk_get_depth

```
int pk_get_depth(Pk_drawable *drawable);
```

Returns the pixel size in use on the drawable.

pk_get_memspace

```
int pk_get_memspace(Pk_drawable *drawable);
```

Returns the memory space in use for the drawable. Memory space designations are board specific and described in the sections 4.3 and 5.3.

pk_set_cliprect

```
void pk_set_cliprect(Pk_drawable *drawable, int left, int top, int  
right, int bottom);
```

Sets the clipping rectangle for the drawable to the bounds *left*, *top*, *right* and *bottom*.

pk_make_color

```
unsigned int pk_make_color(Pk_drawable *drawable, int r, int g, int b);
```

Returns the pixel value to use for the red/green/blue value *r*, *g*, *b* with the input drawable. Red, green and blue values should be in the range 0 to 65535.

pk_extract_color

```
void pk_extract_color(Pk_drawable *drawable, unsigned int pixel, short *r, short *g, short *b);
```

Returns the *rgb* value assigned to the *pixel* value for the input drawable..

pk_read_pixel

```
int pk_read_pixel(Pk_drawable *drawable, int x, int y, unsigned int *color);
```

Returns the value of the pixel at the location (*x*, *y*) in the input drawable in the integer pointed to by *color*. Returns 0 on error.

pk_draw_pixel

```
void pk_draw_pixel(Pk_drawable *drawable, int x, int y, unsigned int color);
```

Draws a pixel of the input *color* at location (*x*, *y*).

pk_draw_line

```
void pk_draw_line(Pk_drawable *drawable, int x0, int y0, int x1, int y1, unsigned int color);
```

Draws a line of the input *color* from (*x0*, *y0*) to (*x1*, *y1*). The pixels at (*x0*, *y0*) and (*x1*, *y1*) will both be drawn.

pk_draw_triangle

```
void pk_draw_triangle(Pk_drawable *drawable, int x0, int y0, int x1, int y1, int x2, int y2, unsigned int color);
```

Draws a triangle of the input *color* with vertices (*x0*, *y0*), (*x1*, *y1*), and (*x2*, *y2*).

pk_draw_rectangle

```
void pk_draw_rectangle(Pk_drawable *drawable, int left, int top, int right, int bottom, unsigned int color);
```

Draws a rectangle of the input *color* with bounds *left*, *top*, *right*, and *bottom*.

pk_draw_polygon

```
void pk_draw_polygon(Pk_drawable *drawable, int npoints, short *x, short *y, unsigned int color);
```

Draws a polygon of the input *color* with *npoints* vertices; the x-coordinates of the vertices are in the array *x*, the y-coordinates of the vertices are in the array *y*.

pk_draw_polyline

```
void pk_draw_polyline(Pk_drawable *drawable, int npoints, short *x, short *y, unsigned int color);
```

Draws a polyline of the input *color* between *npoints* points; the x-coordinates of the points are in the array *x*, the y-coordinates of the points are in the array *y*.

pk_draw_lineSegs

```
void pk_draw_lineSegs(Pk_drawable *drawable, int nsegs, short *x, short *y, unsigned int color);
```

Draws *nsegs* line segments of the input *color*. The $2 * nsegs$ x-coordinates of the segment endpoints are in the array *x*, the $2 * nsegs$ y-coordinates of the segment endpoints are in the array *y*.

pk_draw_circle

```
void pk_draw_circle(Pk_drawable *drawable, int x, int y, int radius, unsigned int color);
```

Draws a circle of the input *color* with radius *radius* and center (*x*, *y*).

pk_draw_ellipse

```
void pk_draw_ellipse(Pk_drawable *drawable, int x, int y, int width, int height, unsigned int color);
```

Draws an ellipse of the input *color* with horizontal axis of width *width* and vertical axis of height *height* and center (*x*, *y*).

pk_draw_arc

```
void pk_draw_arc(Pk_drawable *drawable, int x, int y, int width, int height, int theta, int arc, unsigned int color);
```

The starting angle, *theta*, is measured from the center of the right side of the enclosing rectangle, is expressed in degrees, and is treated modulo 360. Positive angles are in the clockwise direction. The *arc* extent specifies the number of degrees (positive or negative) spanned by the arc; if the absolute value of *arc* is greater than or equal to 360, the entire ellipse is drawn.

pk_draw_piearc

```
void pk_draw_piearc(Pk_drawable *drawable, int x, int y, int width, int height, int theta, int arc, unsigned int color);
```

Draws an arc, with parameters as for **pk_draw_arc**, and connects the endpoints of the arc to the center (*x*, *y*).

pk_fill_triangle

```
void pk_fill_triangle(Pk_drawable *drawable, int x0, int y0, int x1, int y1, int x2, int y2, unsigned int color);
```

Draws a filled triangle with vertices (*x0*, *y0*), (*x1*, *y1*), and (*x2*, *y2*) with the input *color*.

pk_fill_rectangle

```
void pk_fill_rectangle(Pk_drawable *drawable, int left, int top, int right, int bottom, unsigned int color);
```

Draws a filled rectangle with bounds *left*, *top*, *right*, and *bottom* with the input *color*.

pk_fill_polygon

```
void pk_fill_polygon(Pk_drawable *drawable, int npoints, short *x, short *y, unsigned int color);
```

Draws a filled polygon of *npoints* vertices with the input *color*; the *npoints* x-coordinates of the vertices are in the array *x*, the *npoints* y-coordinates of the vertices are in the array *y*.

pk_fill_circle

```
void pk_fill_circle(Pk_drawable *drawable, int x, int y, int radius, unsigned int color);
```

Draws a filled circle with radius *radius* and center (*x*, *y*) with the input *color*.

pk_fill_ellipse

```
void pk_fill_ellipse(Pk_drawable *drawable, int x, int y, int width,  
                     int height, unsigned int color);
```

Draws a filled ellipse with horizontal axis of width *width* and vertical axis of height *height* and center (*x,y*) with the input *color*.

pk_fill_piearc

```
void pk_fill_piearc(Pk_drawable *drawable, int x, int y, int width, int  
                     height, int theta, int arc, unsigned int color);
```

Draws a filled pie arc, with parameters as for **pk_draw_piearc**, with the input color..

pk_create_colormap

```
Pk_colormap *pk_create_colormap(Pk_drawable *drawable);
```

Returns a pointer to a colormap that has been created and associated with the input drawable. Returns NULL on failure.

pk_get_colormap

```
Pk_colormap *pk_get_colormap(Pk_drawable *drawable);
```

Returns a pointer to the colormap associated with a drawable.

pk_destroy_colormap

```
void pk_destroy_colormap(Pk_drawable *drawable);
```

Destroys the colormap; the pointer should not be used after this call.

pk_draw_char

```
int pk_draw_char(Pk_drawable *drawable, Pk_font *font, int x, int y,  
                 unsigned int ch, unsigned int color);
```

Draws character *ch*, in font *font*, at location (*x, y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_draw_text

```
int pk_draw_text(Pk_drawable *drawable, Pk_font *font, int x, int y,  
                 char *buf, int len, unsigned int color);
```

Draws *len* characters (from an 8-bit character set) from the buffer *buf*, in font *font*, at location (*x*, *y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_draw_text16

```
int pk_draw_text16(Pk_drawable *drawable, Pk_font *font, int x, int y,  
    short *buf, int len, unsigned int color);
```

Draws *len* characters (from a 16-bit character set) from the buffer *buf*, in font *font*, at location (*x*, *y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_draw_text32

```
int pk_draw_text32(Pk_drawable *drawable, Pk_font *font, int x, int y,  
    int *buf, int len, unsigned int color);
```

Draws *len* characters (from a 32-bit character set) from the buffer *buf*, in font *font*, at location (*x*, *y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_draw_string

```
int pk_draw_string(Pk_drawable *drawable, Pk_font *font, int x, int y,  
    char *str, unsigned int color);
```

Draws characters (from an 8-bit character set) from the null-terminated string *str*, in font *font*, at location (*x*, *y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_draw_string16

```
int pk_draw_string16(Pk_drawable *drawable, Pk_font *font, int x, int  
    y, short *str, unsigned int color);
```

Draws characters (from a 16-bit character set) from the null-terminated string *str*, in font *font*, at location (*x*, *y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_draw_string32

```
int pk_draw_string32(Pk_drawable *drawable, Pk_font *font, int x, int  
    y, int *str, unsigned int color);
```

Draws characters (from a 32-bit character set) from the null-terminated string *str*, in font *font*, at location (*x*, *y*) using pixel value *color* as determined by the colormap of the drawable. Returns 0 on error.

pk_bitblt

```
void pk_bitblt(Pk_drawable *src, int sx, int sy, Pk_drawable *dest, int
dx, int dy, int width, int height, int op);
```

Applies the logical operation *op* to pixels from the *src* and *dest* drawables and places the result in the *dest* drawable. For *i* from 0 to *width* – 1, *j* from 0 to *height* – 1, take the pixel value *S* at location (*sx* + *i*, *sy* + *j*) in the *src* drawable, combine it with pixel value *D* at location (*dx* + *i*, *dy* + *j*) in the *dest* drawable, getting the result as shown in the following table and placing it at location (*dx* + *i*, *dy* + *j*) in the *dest* drawable. If any of the locations are out of range for the drawable referenced, nothing is done to the destination drawable for that (*i*, *j*).

Constant name <i>op</i>	Destination pixel value
PK_ROP_CLEAR	all 0's
PK_ROP_NOR	$\sim(S \mid D)$
PK_ROP_ANDINV	$S \& \sim D$
PK_ROP_COPYINV	$\sim S$
PK_ROP_INVAND	$\sim S \& D$
PK_ROP_INV	$\sim D$
PK_ROP_XOR	$S \wedge D$
PK_ROP_NAND	$\sim(S \& D)$
PK_ROP_AND	$S \& D$
PK_ROP_EQUIV	$\sim(S \wedge D)$
PK_ROP_NOP	D
PK_ROP_ORINV	$S \mid \sim D$
PK_ROP_COPY	S
PK_ROP_INVOR	$\sim S \mid D$
PK_ROP_OR	$S \mid D$
PK_ROP_SET	all 1's

pk_block_copy

```
void pk_block_copy(Pk_drawable *src, int sx, int sy, Pk_drawable *dest,
int dx, int dy, int width, int height);
```

Copies pixels from the *src* drawable and places the result in the *dest* drawable. For *i* from 0 to *width* – 1, *j* from 0 to *height* – 1, put the pixel value *S* from location (*sx* + *i*, *sy* + *j*) in the *src* drawable at location (*dx* + *i*, *dy* + *j*) in the *dest* drawable. If any of the locations are out of range for the drawable referenced, nothing is done to the destination drawable for that (*i*, *j*).

pk_shaded_line

```
void pk_shaded_line(Pk_drawable *drawable, int x0, int y0, int r0, int
g0, int b0, int x1, int y1, int r1, int g1, int b1);
```

Draws a line from $(x0, y0)$ to $(x1, y1)$ in the drawable, shading it from rgb color $(r0, g0, b0)$ to color $(r1, g1, b1)$.

pk_shaded_triangle

```
void pk_shaded_triangle(Pk_drawable *drawable, int x0, int y0, int r0,
int g0, int b0, int x1, int y1, int r1, int g1, int b1, int x2,
int y2, int r2, int g2, int b2);
```

Draws a shaded triangle, with vertex $(x0, y0)$ having color $(r0, g0, b0)$, vertex $(x1, y1)$ having color $(r1, g1, b1)$ and vertex $(x2, y2)$ having color $(r2, g2, b2)$.

pk_read_span

```
int pk_read_span(Pk_drawable *drawable, int x, int y, int width, void
*buf);
```

Reads a horizontal span of *width* pixels, starting at location (x, y) from the drawable into the buffer *buf*. The type of *buf* should be either `unsigned char`, `unsigned short` or `unsigned int` depending on the pixel size. For pixel sizes eight bits or less the type should be `unsigned char`, for nine to 16 bits, the type should be `unsigned short`, and for 17 to 32 bits, `unsigned int`. Returns 0 on error.

pk_write_span

```
int pk_write_span(Pk_drawable *drawable, int x, int y, int width, void
*buf);
```

Writes a horizontal span of *width* pixels, starting at location (x, y) into the drawable from the buffer *buf*. See **pk_read_span** for a description of the type of the array *buf*. Returns 0 on error.

pk_read_pixmap

```
int pk_read_pixmap(Pk_drawable *drawable, int x, int y, int width, int
height, int pitch, void *buf);
```

Reads an area of *width* by *height* pixels with upper left corner at location (x, y) in the drawable, and copies it into a buffer *buf* with pitch *pitch*. The pitch of the buffer is the increment to use for the width of a row in the *buf* whenever the *y* coordinate is incremented when copying the pixels. The buffer *buf* is logically a two-dimensional array of a data type the same size as a pixel; the array may have rows wider than *width*, and must have at least *height* rows. See **pk_read_span** for a description of the type of the array *buf*. Returns 0 on error.

pk_write_pixmap

```
int pk_write_pixmap(Pk_drawable *drawable, int x, int y, int width, int height, int pitch, void *buf);
```

Writes an area of *width* by *height* pixels with upper left corner at location *(x, y)* into the drawable, getting the data from a buffer *buf* with pitch *pitch*. The pitch of the buffer is the increment to use for the width of a row in the *buf* whenever the *y* coordinate is incremented when copying the pixels. The buffer *buf* is logically a two-dimensional array of a data type the same size as a pixel; the array may have rows wider than *width*, and must have at least *height* rows. See **pk_read_span** for a description of the type of the array *buf*. Returns 0 on error.

pk_flush

```
void pk_flush(Pk_drawable *drawable);
```

Flushes any caches associated with the drawable. Pixels written to a drawable may not appear on the screen until after **pk_flush** has been called.

3.5 Pk_font Methods

pk_destroy_font

```
void pk_destroy_font(Pk_font *font);
```

Destroys a font object that was originally created by **pk_create_t1_font**, **pk_create_t1_font_mem**, or **pk_create_truetype_font**.

pk_get_ascent

```
int pk_get_ascent(Pk_font *font);
```

Returns the distance in pixels from the base line to the topmost pixel drawn for any character in the font..

pk_get_descent

```
int pk_get_descent(Pk_font *font);
```

Returns the distance in pixels from the base line to the lowest pixel drawn for any character in the font.

pk_get_spacing

```
int pk_get_spacing(Pk_font *font);
```

Returns the vertical spacing for the font..

pk_glyph_info

```
int pk_glyph_info(Pk_font *font, unsigned int ch, Pk_glyph_info *info);
```

Returns a variety of information about the glyph for the character *ch* in the font. See the description of **Pk_glyph_info**.

4 The libVFX library

This library implements specializations of the **libPK** library data structures and methods for the VFX-M. These structures and methods should generally be accessed using the functions from the **libPK** library. The **libWs** library, or for the time being, function **pk_create_board**, can be used to associate the appropriate methods with the **Pk_board** structure.

This section describes how to configure and test your VFX-M. Configuration procedures vary depending on your processor board and operating system. Information is also given about the pixel formats available for this graphics board, the memory spaces that may be used for creating drawables, and the kinds of information about this board that may be obtained from **pk_get_boardinfo**.

No documentation is provided for most of the methods in **libVFX**. If you wish to program at that level, you must be prepared to spend time reading source code and the *VFX-M Hardware Manual*. In addition to the VFX-M methods used by the **libPK** library, a full set of macros for reading and writing the registers of the Imagine 128 Number 9 graphics chip and the TVP3030 DAC is provided. See the header file **vfx.h** in the Peritek GLP distribution. Caution must be exercised in mixing direct hardware access with **libPK** calls. Contact Peritek for more information.

4.1 Configuration and test

Special configuration procedures are described below. If there is no heading for your processor board and operating system type, then you should be able to run the Peritek demos (see section 1.3, **Getting started**) without needing a special configuration step.

VxWorks, MVME2604

There are two problems that must be solved with this configuration. First, the default VxWorks kernel does not provide enough CPU-PCI memory space to accommodate the device. In order to make this memory space available, the **sysPhysMemDesc** table in **sysLib.c** must be modified and your VxWorks kernel rebuilt. A sample file for systems using the non-EXTENDED_VME memory map is provided in **vxworks/sysLib.c**. You may use this file instead of the **sysLib.c** file in your Tornado1.0.1 mv2604 BSP 1/4 directory, after checking to make sure that the only differences between the two files are those in the **sysPhysMemDesc** table. If there are other differences, if you are using the EXTENDED_VME configuration, or if you are having trouble recompiling your kernel, please contact Peritek for assistance.

Second, the MVME 2604 does not configure PCI bus devices in firmware, so a special function must be run to do this on boot-up before any other routines are run. Here is a Tornado shell script that calls the **libVFX** function that does this. This script can be found in the distribution as **vxworks/vfx_mv2604.scp**. The object file can be found as **obj/vxw-mv2604/libglp.o**. To avoid problems with path names, copy these files to a test directory before loading the script, which uses only local file names. Here is the contents of the script **vfx_mv2604.scp**.

```
# This script programs the header of the VFX-M on the MV2604,
# since the OS and firmware fail to do so. It must be run
# on power up, before any other VFX-M activity.

ld < libglp.o

vfxbus = 0
vfxslot = 16
pcimembase = 0x03000000
pcioibase = 0x01080000
interrupt_line = 0x19

vfx_config(0, vfxbus, vfxslot, pcimembase, pcioibase, interrupt_line)
```

After running the script, use the Tornado functions **pciDeviceShow** and **pciHeaderShow** to see if the VFX-M has been configured properly. Here is a screen dump of the correct output for **pciDeviceShow**.

```
-> pciDeviceShow(0)
Scanning function 0 of each PCI device on bus 0
Using configuration mechanism 1
bus      device      function  vendorID  deviceID  class
00000000  00000000  00000000  00001057  00004801  00060000
00000000  0000000b  00000000  000010ad  00000565  00060100
00000000  0000000c  00000000  00001000  00000003  00010000
00000000  0000000d  00000000  000010e3  00000000  00068000
00000000  0000000e  00000000  00001011  00000009  00020000
00000000  00000010  00000000  0000105d  00002339  00030000
value = 0 = 0x0
```

The Peritek VFX-M graphics card with vendor ID 0x105d, device ID 0x2339 is found at bus 0, device 0x10 (16), func 0. If the device location is not 16, you will need to specify the actual device location as the third argument to **vfx_configure** in **vfx_mv2604.scp**. If no device shows the correct vendorID and deviceID, you have a hardware or operating system installation problem, please contact Peritek for assistance.

Here is the correct output for **pciHeaderShow** of the VFX-M after the base address registers have been programmed with the script **vfx_mv2604.scp**:

```
-> pciHeaderShow(0, 16, 0)
vendor ID =          0x105d
device ID =          0x2339
command register =  0x0003
status register =   0x0280
revision ID =        0x02
class code =         0x03
sub class code =    0x00
programming interface = 0x00
cache line =         0x00
latency time =       0x00
header type =        0x00
BIST =               0x00
base address 0 =     0x04000008
base address 1 =     0x06000008
```

```
base address 2 =          0x03400000
base address 3 =          0x03800000
base address 4 =          0x03120000
base address 5 =          0x01080001
cardBus CIS pointer =    0x00000000
sub system vendor ID =   0x10f0
sub system ID =          0x0000
expansion ROM base address = 0x03100001
interrupt line =          0x19
interrupt pin =           0x01
min Grant =               0x00
max Latency =             0x00
value = 0 = 0x0
->
```

If the base address registers and the interrupt line are not set to the above values, then the PCI header on the first device was not initialized. Please contact Peritek for assistance. If it is correct, you are ready to start displaying the demos.

Running demos on VxWorks PowerPC systems

After performing any processor-specific configuration, copy **obj/vxw-ppc/libglp.o** from the distribution into your test directory, if you have not already copied a processor-specific version of **libglp.o**. Then copy the following files into your test directory.

obj/vxw-ppc/pktestpat
obj/vxw-ppc/pkflag
obj/vxw-ppc/pkdemo
vxworks/vcqm_demo.scp

Either copy or link the entire **fonts** subtree into the directory as well. The demo programs use font names that assume the fonts directory is a subdirectory of the current directory.

Then you may run the Tornado shell script **vfx_demo.scp**,

```
< vfx_demo.scp
```

which is shown below:

```
# vfxbus and vfxslot must be set to the location of the VFX-M
# on the PCI bus

vfxbus = 0
vfxslot = 16

ld < libglp.o
ld < pktestpat

# Demo of board on single VFX-M running at different resolutions
# The last argument controls which demo to run.

board = pktestpat(vfxbus, vfxslot, 0, 640, 640, 66000, 8, 0)
board = pktestpat(vfxbus, vfxslot, 0, 800, 600, 66000, 8, 0)
```

```
board = pktestpat(vfxbus, vfxslot, 0, 1024, 768, 66000, 8, 0)
board = pktestpat(vfxbus, vfxslot, 0, 1280, 1024, 60000, 8, 1)
board = pktestpat(vfxbus, vfxslot, 0, 1600, 1200, 60000, 8, 0)

# Other graphics functions that operated on the board pointer returned
# could be inserted here. pkflag is an example demo.

ld <pkflag
pkflag(board, 1280, 1024, 60000, 8)

# The following demo uses libMesa to demonstrate shaded drawing
# using OpenGL
ld <pkdemo

pkdemo(vfxbus, vfxslot, 0, 1280, 1024)
```

You should first see test patterns in a variety of pixel formats, then a flag with font display, then shaded teapots. To examine any of the patterns more closely, just edit the script to execute only the pattern you are interested in. If there are any failures, please contact Peritek for assistance.

4.2 Pixel format

The VFX-M provides a rich set of pixel formats. The following table shows the constant names designating the formats; include the header file **vxh.h** for the definitions. These constants may be used as the *pixformat* argument in calls to **pk_set_framebuffer**, **pk_create_drawable**, and **Pk_pixelinfo**. The latter function will give you more information about a given pixel format, see the description of **Pk_pixelinfo** in section 3.1.

pixformat	Description
VFX_PIXFORMAT_8BIT	Eight bit pixel
VFX_PIXFORMAT_16BIT_1555	16-bit pixel, 1 overlay, 5 red, 5 green, 5 blue
VFX_PIXFORMAT_16BIT_4444	16-bit pixel, 4 red, 4 green, 4 blue, 4 overlay
VFX_PIXFORMAT_16BIT_565	16-bit pixel, 5 red, 6 green, 5 blue
VFX_PIXFORMAT_16BIT_664	16-bit pixel, 6 red, 6 green, 4 blue
VFX_PIXFORMAT_24BIT_1888	24-bit pixel, 8 bits each color, 5 pixels per 16 bytes, 1 byte unused
VFX_PIXFORMAT_24BIT_888	24-bit pixel, 8 bits each color, 16 pixels per 48 bytes, none unused
VFX_PIXFORMAT_32BIT	32 bit pixel, 8 unused, 8 red, 8 green, 8 blue
VFX_PIXFORMAT_32BIT_8888	32 bit pixel, 8 overlay, 8 red, 8 green, 8 blue

The colormap is used for looking up overlay colors in 16BIT_4444, 16BIT_1555, and 32BIT_8888 formats. The colormap is used for gamma correction in 16BIT_664, 16BIT_565, 24BIT_1888, 24BIT_888, and 32BIT modes. The colormap is used for color lookup in 8BIT mode.

4.3 Memory spaces

The VFX-M has three memory spaces; two of them can be used to create off-screen drawables.

<i>memspace</i> constant	Usage
VFX_MEMSPACE_VMEM	Used for creating drawables in video memory.
VFX_MEMSPACE_DMEM	Used for creating drawables in off screen DRAM.
VFX_MEMSPACE_MMEM	Used to indicate mask memory. Drawable can not be created in mask memory.

See the *VFX-M Hardware Manual* for more information.

4.4 Board info

The following constants, defined in **vfxf.h**, may be used as the *info* parameter with **pk_get_boardinfo** to get more information about your VFX-M.

Board info constant	Returned in <i>value</i>	Notes
VFX_BOARDINFO_MW0SIZE	Size of memory window zero.	Likely to be 16 Mbytes, available for user applications.
VFX_BOARDINFO_MW1SIZE	Size of memory window one.	Likely to be 32 Mbytes, used by the libVFX rendering functions;
VFX_BOARDINFO_XYWASIZE	Size of XY window A.	Likely to be 4 Mbytes, used by the libVFX rendering functions;
VFX_BOARDINFO_XYWBSIZE	Size of XY window B.	Likely to be 4 Mbytes, not usable.
VFX_BOARDINFO_RBASESIZE	Size of control register space.	Should be 64Kbytes.
VFX_BOARDINFO_VMEMSIZE	Size of installed video memory.	Usually 8 Mbytes.
VFX_BOARDINFO_DMEMSIZE	Size of installed DRAM.	Usually 64 Mbytes
VFX_BOARDINFO_MMEMSIZE	Size of installed mask buffer DRAM.	Usually 2 Mbytes.
VFX_BOARDINFO_VMEMFREE	Video memory not currently allocated to drawables.	
VFX_BOARDINFO_DMEMFREE	DRAM not currently allocated to drawables.	
VFX_BOARDINFO_MMEMFREE	Mask memory DRAM not	

	currently allocated for fonts or patterns.	
--	--	--

Memory window zero is the one available for use by user applications. Care should be taken when using this window with DRAM that the bank select control bit is set and restored properly. XY window B is not fully implemented on the Number Nine i128 series 2 part, and is therefore not usable. See the *VFX-M Hardware Manual* for more information about memory windows on the i128.

4.5 VFX-M specific functions

vfx_load_timing

```
int vfx_load_timing(Vfx *vfx, double vfreq, double vblank, double
    vfporch, double vsync, double hblank, double hfporch, double
    hsync, int width, int height, int buswidth, int bitsperpixel, int
    pixelformat);
```

Allows more precise control over the setting of video timing. This function can be used after calling **pk_set_framebuffer** or it may be used without calling **pk_set_framebuffer** if care is taken to set up the primary drawable with a call to **pk_create_drawable**. *vfreq* is the vertical refresh frequency in hertz. *vblank* is the vertical blanking interval in seconds. *vfporch* is the interval between the start of blanking and the start of vertical sync in seconds. *vsync* is the duration of vertical sync in seconds. *hblank* is the horizontal blanking interval in seconds. *hfporch* is the interval between the start of blanking and the start of horizontal sync in seconds. *hsync* is the duration of horizontal sync in seconds. *width* is the width of the display in pixels. *height* is the height of the display in scanlines. *buswidth* is the width the video memory bus in bits and should always be 64. *bitsperpixel* is the number of bits per pixel for the requested pixel format. *pixelformat* is the graphics board specific pixel format constant. Returns 0 on error.

vfx_show_timing

```
int vfx_show_timing(double vfreq, double vblank, double vfporch, double
    vsync, double hblank, double hfporch, double hsync, int width,
    int height, int buswidth, int bitsperpixel, int pixelformat)
```

Display the computed timing information and control register values for the requested display timing. It should be used to verify that the VFX-M can adequately support your custom display timing. Parameter values are as for **vfx_show_timing**. Returns 0 on error.

vfx_write_palette

```
void vfx_write_palette(Vfx *board, int index, int r, int g, int b);
```

This provides an alternate method of writing directly to the VFX-M color palette without going through the **Pk_colormap** methods. Care should be taken to use either one or the other of the two mechanisms. It is recommended that you use the **Pk_colormap** methods if at all possible since they are more likely to optimized in the future. **vfx_write_palette** does provide raw access to the color palette for those applications that require it.

5 The libVCQM library

This library implements specializations of the **libPK** library data structures and methods for the VCQ-M. These structures and methods should generally be accessed using the functions from the **libPK** library. The **libWs** library, or for the time being, function **pk_create_board**, can be used to associate the appropriate methods with the **Pk_board** structure.

This section describes how to configure and test your VCQ-M. A VCQ-M has four Cirrus Logic GD7555 graphics controllers; each of these is configured as a separate graphics “board” and may use different resolutions and pixel formats. Configuration procedures vary depending on your processor board and operating system. Information is also given about the pixel formats, the memory spaces that may be used for creating drawables, and the kinds of information about this board that may be obtained from **pk_get_boardinfo**.

In addition to the VCQ-M methods used by the **libPK** library, a full set of macros for reading and writing the registers of the Cirrus Logic GD7555 graphics controllers are provided for those who wish to program the hardware directly. See the header file **vcqm.h** in the Peritek GLP distribution, and the *VCQ-M Hardware Manual*. Caution must be exercised in mixing direct hardware access with **libPK** calls. Contact Peritek for more information.

5.1 Configuration and test

Special configuration procedures are described below. If there is no heading for your processor board and operating system type, then you should be able to run the Peritek demos (see section 1.3, **Getting started**) without needing a special configuration step.

VxWorks, MVME2604

There are two problems that must be solved with this configuration. First, the default VxWorks kernel does not provide enough CPU-PCI memory space to accommodate the device. In order to make this memory space available, the **sysPhysMemDesc** table in **sysLib.c** must be modified and your VxWorks kernel rebuilt. A sample file for systems using the non-EXTENDED_VME memory map is provided in **vxworks/sysLib.c**. You may use this file instead of the **sysLib.c** file in your Tornado1.0.1 mv2604 BSP 1/4 directory, after checking to make sure that the only differences between the two files are those in the **sysPhysMemDesc** table. If there are other differences, if you are using the EXTENDED_VME configuration, or if you are having trouble recompiling your kernel, please contact Peritek for assistance.

Second, the MVME 2604 does not configure PCI bus devices in firmware, so a special function must be run to do this on boot-up before any other routines are run. Here is a Tornado shell script that calls the **libVCQM** function that does this. This script can be found in the distribution as **vxworks/vcqm_mv2604.scp**. The object file can be found as **obj/vxw-mv2604/libglp.o**. To avoid problems with path names, copy these files to a test directory before loading the script, which uses only local file names. Here is the contents of the script **vcqmn_mv2604.scp**.

```
ld <libglp.o
secbus = 1
pcimembase = 0x02000000
pcimemsize = 0x06000000
vcqm_configure(0, 0, 16, &secbus, &pcimembase, &pcimemsize)
```

After running the script, use the Tornado functions **pciDeviceShow** and **pciHeaderShow** to see if the VCQM has been configured properly. Here is a screen dump of the correct output.

```
-> pciDeviceShow(0)

Scanning function 0 of each PCI device on bus 0
Using configuration mechanism 1

bus      device      function  vendorID  deviceID  class
00000000  00000000  00000000  00001057  00004801  00060000
00000000  0000000b  00000000  000010ad  00000565  00060100
00000000  0000000c  00000000  00001000  00000003  00010000
00000000  0000000d  00000000  000010e3  00000000  00068000
00000000  0000000e  00000000  00001011  00000009  00020000
00000000  00000010  00000000  0000104c  0000ac21  00060400
value = 0 = 0x0
```

The Peritek VCQ-M PCI bridge with vendor ID 0x104c, device ID 0xac21 is found at bus 0, device 0x10 (16), func 0. If the device location is not 16, you will need to specify the actual device location as the third argument to **vcqm_configure** in **vcqm_mv2604.scp**. If no device shows the correct vendorID and deviceID, you have a hardware or operating system installation problem, please contact Peritek for assistance.

```
-> pciDeviceShow(1)
Scanning function 0 of each PCI device on bus 1
Using configuration mechanism 1

bus      device      function  vendorID  deviceID  class
00000001  00000000  00000000  00001013  00000040  00030000
00000001  00000001  00000000  00001013  00000040  00030000
00000001  00000002  00000000  00001013  00000040  00030000
00000001  00000003  00000000  00001013  00000040  00030000
00000001  00000004  00000000  00001045  0000c861  000c0310
value = 0 = 0x0
```

The Cirrus devices on the VCQ-M appear as devices on bus 0. If you do not see four devices with vendorID 0x1013, deviceID 0x0040, contact Peritek for assistance.

```
-> pciHeaderShow(1, 0, 0)
vendor ID =                      0x1013
device ID =                      0x0040
command register =                0x0003
status register =                 0x0200
revision ID =                     0x00
class code =                      0x03
```

```
sub class code =          0x00
programming interface = 0x00
cache line =             0x00
latency time =           0x00
header type =            0x00
BIST =                   0x00
base address 0 =         0x02000000
base address 1 =         0x06002000
base address 2 =         0x00000000
base address 3 =         0x00000000
base address 4 =         0x00000000
base address 5 =         0x00000000
cardBus CIS pointer =   0x00000000
sub system vendor ID = 0x0000
sub system ID =          0x0000
expansion ROM base address = 0x000c0000
interrupt line =         0x00
interrupt pin =          0x01
min Grant =              0x00
max Latency =            0x00
value = 0 = 0x0
```

If the base address 0 and base address 1 are not set to the above values, then the PCI header on the first device was not initialized. Please contact Peritek for assistance. If it is correct, you are ready to start displaying the demos.

Running demos on VxWorks PowerPC systems

After performing any processor-specific configuration, copy **obj/vxw-ppc/libglp.o** from the distribution into your test directory, if you have not already copied a processor-specific version of **libglp.o**. Then copy the following files into your test directory.

obj/vxw-ppc/pktestpat
obj/vxw-ppc/pkmulti
obj/vxw-ppc/pkdemo
vxworks/vcqmdemo.scp

Either copy or link the entire **fonts** subtree into the directory as well. The demo programs use font names that assume the fonts directory is a subdirectory of the current directory. Then you may run the Tornado shell script **vcqmdemo.scp**,

```
< vcqmdemo.scp
```

which is shown below:

```
# Demos running on all four Cirrus chip sets of a single VCQ-M.
# A single VCQ-M functions as four graphics boards.

# PCI secondary bus number for VCQ-M depends on system configuration
# Correct value of vcqbus for MV2604 with one VCQ-M:

vcqbus = 1
```

```
ld < libglp.o
ld < pktestpat

# Demo of four boards on single VCQ-M running at different resolutions
# The last argument controls which demo to run.

board0 = pktestpat(vcqbus, 0, 0, 640, 640, 66000, 8, 0)
board1 = pktestpat(vcqbus, 1, 0, 800, 600, 66000, 8, 0)
board2 = pktestpat(vcqbus, 2, 0, 1024, 768, 66000, 8, 0)
board3 = pktestpat(vcqbus, 3, 0, 512, 512, 66000, 8, 1)

# Other graphics functions that operated on the board pointers
# returned
# could be inserted here.

# Demo of a single program running on all four boards
# (default resolution 1024 x 768, pixformat 8, clock frequency 66000)

ld < pkmulti

# The variable bus1 is a global in pkmulti, be sure to load the pkmulti
# object file before setting bus1 to the correct value for your system.
# To alter the resolution or the pixformat, you may also set the
# global variables in pkmulti.c from the shell (commented out here).
# To run pkmulti for two VCQ-Ms in a single system, you will also
# need to set bus2, as well as changing the argument of pkmulti to
# the number of displays you will actually be running..

bus0 = vcqbus
# width = 640
# height = 480
# clockfreq = 66000
# pixformat = 24

pkmulti(4)

# The following demo uses libPKMesa to demonstrate shaded drawing
# using OpenGL
ld < pkdemo

pkdemo(vcqbus, 0, 0, 640, 480)
pkdemo(vcqbus, 1, 0, 640, 480)
pkdemo(vcqbus, 2, 0, 640, 480)
pkdemo(vcqbus, 3, 0, 640, 480)
```

You should first see test patterns on all four screens, then a flag and font displays, then shaded teapots. To examine any of the patterns more closely, just edit the script to execute only the pattern you are interested in. If there are any failures, please contact Peritek for assistance.

5.2 Pixel format

The following table shows the constant names designating the VCQ-M pixel formats; include the header file **vcqm.h** for the definitions. These constants may be used as the *pixformat* argument in calls to **pk_set_frame_buffer**, **pk_create_drawable**, and **pk_get_pixelinfo**. The latter function will give you more information about a given pixel format, see the description of **Pk_pixelinfo** in section 3.1.

<i>pixformat</i>	Description
VCQM_PIXFORMAT_8BIT	8-bit pixel
VCQM_PIXFORMAT_8BIT_GREY	8-bit pixel, grey scale
VCQM_PIXFORMAT_16BIT	16-bit pixel, 5 red, 6 green, 5 blue
VCQM_PIXFORMAT_16BIT_555	16-bit pixel, 1 unused, 5 red, 5 green, 5 blue
VCQM_PIXFORMAT_24BIT	24-bit pixel, 8 red, 8 green, 8 blue

5.3 Memory spaces

The VCQ-M has only a single memory space per Cirrus Logic GD7555 graphics controller, though each of the four controllers is mapped into a different region of the processor's address space. Either of the memory space designators **PK_MEMSPACE_VMEM** or **VCQM_MEMSPACE_MEM** can be used in calls to **pk_create_drawable**.

5.4 Board info

The following constants, defined in **vcqm.h**, may be used as the *info* parameter with **pk_get_boardinfo** to get more information about any of the “boards” (Cirrus Logic GD7555 graphics controllers) on your VCQ-M.

Board info constant	Returned in value	Notes
VCQM_BOARDINFO_MWSIZE	Size of the display memory window	Always 4Mbytes.
VCQM_BOARDINFO_MEMSIZE	Size of display memory.	Probed by pk_create_board , generally 4Mbytes.
VCQM_BOARDINFO_MEMFREE	The number of bytes of display memory not allocated to drawables at the present time	

5.5 VCQ-M specific functions

vcqm_access_registers

```
void vcqm_access_registers(Vcqm *vcqm);
```

Provides an interactive tool to view and modify the contents of the VCQ-M control registers. This is used primarily for development of libVCQM.

vcqm_configure

```
int vcqm_configure(char *devname, int bus, int dev, int *secbus, int
                   *pcimembase, int *pcimemszie)
```

Configures the VCQ-M graphics controller in systems where the VCQ-M is not configured by the system BIOS firmware or operating system. The parameters *devname*, *bus* and *dev* identify the VCQ-M on-board TI PCI2031 PCI-PCI bridge chip. The parameters *secbus*, *pcimembase* and *pcimemszie* are pointers to the secondary bus number, the base address of unallocated PCI memory, and the size of unallocated PCI memory, respectively. Upon return these variables are updated to reflect the amount of PCI address space consumed by the VCQ-M and the allocation of a single bus number. Returns 0 on error.

vcqm_load_timing

```
int vcqm_load_timing(Vcqm *vcqm, double vfreq, double vblank, double
                      vfporch, double vsync, double hblank, double hfporch, double
                      hsync, int width, int height, int buswidth, int bitsperpixel, int
                      pixelformat);
```

Allows more precise control over the setting of video timing. This function can be used after calling **pk_set_framebuffer** or it may be used without calling **pk_set_framebuffer** if care is taken to setup the primary drawable with a call to **pk_create_drawable**. *vfreq* is the vertical refresh frequency in hertz. *vblank* is the vertical blanking interval in seconds. *vfporch* is the interval between the start of blanking and the start of vertical sync in seconds. *vsync* is the duration of vertical sync in seconds. *hblank* is the horizontal blanking interval in seconds. *hfporch* is the interval between the start of blanking and the start of horizontal sync in seconds. *hsync* is the duration of horizontal sync in seconds. *width* is the width of the display in pixels. *height* is the height of the display in scanlines. *buswidth* is the width the video memory bus in bits and should always be 64. *bitsperpixel* is the number of bits per pixel for the requested pixel format. *pixelformat* is the graphics board specific pixel format constant. Returns 0 on error.

vcqm_show_timing

```
int vcqm_show_timing(double vfreq, double vblank, double vfporch,
                     double vsync, double hblank, double hfporch, double hsync, int
                     width, int height, int buswidth, int bitsperpixel, int
                     pixelformat)
```

Display the computed timing information and control register values for the requested display timing. It should be used to verify that the VCQ-M can adequately

support your custom display timing. Parameter values are as for **vcqm_load_timing**. Returns 0 on error.

vcqm_write_palette

```
void vcqm_write_palette(Vcqm *board, int index, int r, int g, int b);
```

This provides an alternate method of writing directly to the VCQ-M color palette without going through the **Pk_colormap** methods. Care should be taken to use either one or the other of the two mechanisms. It is recommended that you use the **Pk_colormap** methods if at all possible since they are more likely to optimized in the future. **vcqm_write_palette** does provide raw access to the color palette for those applications that require it.

vcqm_write_cursor_color

```
void vcqm_write_cursor_color(Vcqm *vcqm, int index, int r, int g, int b);
```

Provides an alternate method of writing directly to the VCQ-M cursor color palette. The VCQ-M hardware cursor is a two bit cursor. The cursor values are 0 for transparent, 1 for exclusive-or of normal video data, 2 for the background cursor color, and 3 for the foreground cursor color. Only cursor colors 2 and 3 can be used as the *index* and modified with the function.

The preferred mechanism is to use the **Pk_colormap** associated with the cursor.

```
Pk_drawable *cursor_drawable = pk_get_cursordrawable(board);
Pk_colormap *cursor_colormap = pk_get_colormap(cursor_drawable);
unsigned int background =
    pk_allocate_color(cursor_colormap, bred, bgreen, bblue);
unsigned int foreground =
    pk_allocate_color(cursor_colormap, fred, fgreen, fblue);
```

The color numbers *background* and *foreground* can then be used for drawing on the cursor drawable. Currently these will be the values 2 and 3 but it is preferable not to depend upon this fact in order to aid compatibility with future revisions of **libVCQM**.

6 The libPKMesa library

The **Mesa 3-D Graphics Library** is a publicly available software package that implements the OpenGL API. Peritek has ported this package to our boards to support applications using the OpenGL API. See the Mesa 3-D website, <http://www.mesa3d.org> for a description of this library and documentation of the data structures and functions available from the package. Their web page includes many pointers to other information about the OpenGL API.

6.1 Data structures

See Mesa documentation for further details.

PKMesaVisual

Peritek specific container for **GLvisual** objects.

PKMesaBuffer

Peritek specific container for **GLframebuffer** objects.

PKMesaContext

Peritek specific container for **GLcontext** objects.

6.2 Functions

PKMesaCreateVisual

PKMesaDestroyVisual

```
PKMesaVisual PKMesaCreateVisual(GLboolean rgb_mode, GLboolean dbFlag,
                                GLint depthSize, GLint stencilSize, GLint accumSize;

void PKMesaDestroyVisual(PKMesaVisual visual);
```

It is necessary to call **PKMesaCreateVisual** to construct a **PKMesaVisual** object prior to calling **PKMesaCreateContext**. When the associated context is destroyed **PKMesaDestroyVisual** can be called to destroy the **PKMesaVisual** object and release its associated resources. *dbFlag* should be `GL_TRUE` if double buffering is required and `GL_FALSE` otherwise. *rgb_mode* should be `GL_TRUE` for direct RGB pixel formats and `GL_FALSE` for color indexed pixel formats. *depthSize* should be 16 if needed and 0 otherwise. *stencilSize* should be 8 if needed and 0 otherwise. *accumSize* should be 16 if needed and 0 otherwise.

PKMesaCreateBuffer **PKMesaDestroyBuffer**

```
PKMesaBuffer PKMesaCreateBuffer(PKMesaVisual visual, Pk_drawable *buf);  
void PKMesaDestroyBuffer(PKMesaBuffer buffer);
```

A **PKMesaBuffer** object is returned by **PKMesaCreateBuffer**. When this object is no longer needed it can be destroyed with **PKMesaDestroyBuffer**. *visual* is a **PKMesaVisual** object created previously with **PKMesaCreateVisual**. *buf* is a libPK drawable object either from **pk_get_drawable** or **pk_create_drawable**.

PKMesaCreateContext **PKMesaDestroyContext**

```
PKMesaContext PKMesaCreateContext(PKMesaVisual visual, PKMesaContext  
sharelist);  
void PKMesaDestroyContext(PKMesaContext context);
```

PKMesaCreateContext makes a **PKMesaContext** object; *visual* is a **PKMesaVisual** object that has been previously created with **PKMesaCreateVisual**. *sharelist* should be NULL.

PKMesaMakeCurrent

```
void PKMesaMakeCurrent(PKMesaContext context, PKMesaBuffer buffer);
```

Makes *context* and *buffer* the current GL rendering context. This function must be called prior to calling any of the **libGL** functions.

PKMesaSwapBuffers

```
void PKMesaSwapBuffers(PKMesaBuffer buffer);
```

Swap the front and back buffers.

7 The libGL library

The following OpenGL API functions are provided. For VxWorks these functions are in the file **mesa.o**. Consult the OpenGL references listed below for specifics on the OpenGL API.

OpenGL Programming Guide
 OpenGL Architecture Review Board
 Silicon Graphics, Inc. 1993
 Addison Wesley, Menlo Park, California

OpenGL Reference Manual, Second Edition
 OpenGL Architecture Review Board
 Silicon Graphics, Inc. 1997
 Addison Wesley, Menlo Park, California

OpenGL Programming for the X Window System
 Mark J. Kilgard, 1996
 Addison Wesley, Menlo Park, California

Base OpenGL functions

glAccum	glColor4dv	glEdgeFlagPointer
glAlphaFunc	glColor4f	glEdgeFlagv
glAreTexturesResident	glColor4fv	glEnable
glArrayElement	glColor4i	glEnableClientState
glBegin	glColor4iv	glEnd
glBindTexture	glColor4s	glEndList
glBitmap	glColor4sv	glEvalCoord1d
glBlendFunc	glColor4ub	glEvalCoord1dv
glCallList	glColor4ubv	glEvalCoord1f
glCallLists	glColor4ui	glEvalCoord1fv
glClear	glColor4uiv	glEvalCoord2d
glClearAccum	glColor4us	glEvalCoord2dv
glClearColor	glColor4usv	glEvalCoord2f
glClearDepth	glColorMask	glEvalCoord2fv
glClearIndex	glColorMaterial	glEvalMesh1
glClearStencil	glColorPointer	glEvalMesh2
glClipPlane	glCopyPixels	glEvalPoint1
glColor3b	glCopyTexImage1D	glEvalPoint2
glColor3bv	glCopyTexImage2D	glFeedbackBuffer
glColor3d	glCopyTexSubImage1D	glFinish
glColor3dv	glCopyTexSubImage2D	glFlush
glColor3f	glCopyTexSubImage3D	glFogf
glColor3fv	glCullFace	glFogfv
glColor3i	glDeleteLists	glFogi
glColor3iv	glDeleteTextures	glFogiv
glColor3s	glDepthFunc	glFrontFace
glColor3sv	glDepthMask	glFrustum
glColor3ub	glDepthRange	glGenLists
glColor3ubv	glDisable	glGenTextures
glColor3ui	glDisableClientState	glGetBooleanv
glColor3uiv	glDrawArrays	glGetClipPlane
glColor3us	glDrawBuffer	glGetDoublev
glColor3usv	glDrawElements	glGetError
glColor4b	glDrawPixels	glGetFloatv
glColor4bv	glDrawRangeElements	glGetIntegerv
glColor4d	glEdgeFlag	glGetLightfv

glGetLightiv	glMap1f	glRasterPos3d
glGetMapdv	glMap2d	glRasterPos3dv
glGetMapfv	glMap2f	glRasterPos3f
glGetMapiv	glMapGrid1d	glRasterPos3fv
glGetMaterialfv	glMapGrid1f	glRasterPos3i
glGetMaterialiv	glMapGrid2d	glRasterPos3iv
glGetPixelMapfv	glMapGrid2f	glRasterPos3s
glGetPixelMapuv	glMaterialf	glRasterPos3sv
glGetPixelMapusv	glMaterialfv	glRasterPos4d
glGetPointerv	glMateriali	glRasterPos4dv
glGetPolygonStipple	glMaterialiv	glRasterPos4f
glGetString	glMatrixMode	glRasterPos4fv
glGetTexEnvfv	glMultMatrixd	glRasterPos4i
glGetTexEnviv	glMultMatrixf	glRasterPos4iv
glGetTexGendv	glNewList	glRasterPos4s
glGetTexGenfv	glNormal3b	glRasterPos4sv
glGetTexGeniv	glNormal3bv	glReadBuffer
glGetTexImage	glNormal3d	glReadPixels
glGetTexLevelParameterfv	glNormal3dv	glRectd
glGetTexLevelParameteriv	glNormal3f	glRectdv
glGetTexParameterfv	glNormal3fv	glRectf
glGetTexParameteriv	glNormal3i	glRectfv
glHint	glNormal3iv	glRecti
glIndexMask	glNormal3s	glRectiv
glIndexPointer	glNormal3sv	glRects
glIndexd	glNormalPointer	glRectsv
glIndexdv	glOrtho	glRenderMode
glIndexf	glPassThrough	glRotated
glIndexfv	glPixelMapfv	glRotatef
glIndexi	glPixelMapuv	glScaled
glIndexiv	glPixelMapusv	glScalef
glIndexs	glPixelStoref	glScissor
glIndexsv	glPixelStorei	glSelectBuffer
glIndexub	glPixelTransferf	glShadeModel
glIndexubv	glPixelTransferi	glStencilFunc
glInitNames	glPixelZoom	glStencilMask
glInterleavedArrays	glPointSize	glStencilOp
glIsEnabled	glPolygonMode	glTexCoord1d
glIsList	glPolygonOffset	glTexCoord1dv
glIsTexture	glPolygonStipple	glTexCoord1f
glLightModelf	glPopAttrib	glTexCoord1fv
glLightModelfv	glPopClientAttrib	glTexCoord1i
glLightModeli	glPopMatrix	glTexCoord1liv
glLightModeliv	glPopName	glTexCoord1s
glLightf	glPrioritizeTextures	glTexCoord1sv
glLightfv	glPushAttrib	glTexCoord2d
glLighti	glPushClientAttrib	glTexCoord2dv
glLightiv	glPushMatrix	glTexCoord2f
glLineStipple	glPushName	glTexCoord2fv
glLineWidth	glRasterPos2d	glTexCoord2i
glListBase	glRasterPos2dv	glTexCoord2iv
glLoadIdentity	glRasterPos2f	glTexCoord2s
glLoadMatrixd	glRasterPos2fv	glTexCoord2sv
glLoadMatrixf	glRasterPos2i	glTexCoord3d
glLoadName	glRasterPos2iv	glTexCoord3dv
glLogicOp	glRasterPos2s	glTexCoord3f
glMap1d	glRasterPos2sv	glTexCoord3fv

glTexCoord3i	glTexGeni	glVertex2sv
glTexCoord3iv	glTexGeniv	glVertex3d
glTexCoord3s	glTexImage1D	glVertex3dv
glTexCoord3sv	glTexImage2D	glVertex3f
glTexCoord4d	glTexImage3D	glVertex3fv
glTexCoord4dv	glTexParameterf	glVertex3i
glTexCoord4f	glTexParameterfv	glVertex3iv
glTexCoord4fv	glTexParameterfi	glVertex3s
glTexCoord4i	glTexParameteriv	glVertex3sv
glTexCoord4iv	glTexSubImage1D	glVertex4d
glTexCoord4s	glTexSubImage2D	glVertex4dv
glTexCoord4sv	glTexSubImage3D	glVertex4f
glTexCoordPointer	glTranslated	glVertex4fv
glTexEnvf	glTranslatef	glVertex4i
glTexEnvfv	glVertex2d	glVertex4iv
glTexEnvi	glVertex2dv	glVertex4s
glTexEnviv	glVertex2f	glVertex4sv
glTexGend	glVertex2fv	glVertexPointer
glTexGendv	glVertex2i	glViewport
glTexGenf	glVertex2iv	
glTexGenfv	glVertex2s	

OpenGL extensions

glAreTexturesResidentEXT	glMultiTexCoord2iEXT
glArrayElementEXT	glMultiTexCoord2ivEXT
glBindTextureEXT	glMultiTexCoord2sEXT
glBlendColorEXT	glMultiTexCoord2svEXT
glBlendEquationEXT	glMultiTexCoord3dEXT
glColorPointerEXT	glMultiTexCoord3dvEXT
glColorSubTableEXT	glMultiTexCoord3fEXT
glColorTableEXT	glMultiTexCoord3fvEXT
glCopyTexSubImage3DEXT	glMultiTexCoord3iEXT
glDeleteTexturesEXT	glMultiTexCoord3ivEXT
glDrawArraysEXT	glMultiTexCoord3sEXT
glEdgeFlagPointerEXT	glMultiTexCoord3svEXT
glGenTexturesEXT	glMultiTexCoord4dEXT
glGetColorTableEXT	glMultiTexCoord4dvEXT
glGetColorTableParameterfvEXT	glMultiTexCoord4fEXT
glGetColorTableParameterivEXT	glMultiTexCoord4fvEXT
glGetPointervEXT	glMultiTexCoord4iEXT
glIndexPointerEXT	glMultiTexCoord4ivEXT
glInterleavedTextureCoordSetsEXT	glMultiTexCoord4sEXT
glIsTextureEXT	glMultiTexCoord4svEXT
glMultiTexCoord1dEXT	glNormalPointerEXT
glMultiTexCoord1dvEXT	glPointParameterfEXT
glMultiTexCoord1fEXT	glPointParameterfvEXT
glMultiTexCoord1fvEXT	glPolygonOffsetEXT
glMultiTexCoord1liEXT	glPrioritizeTexturesEXT
glMultiTexCoord1livEXT	glSelectTextureCoordSetEXT
glMultiTexCoord1lsEXT	glSelectTextureEXT
glMultiTexCoord1lsvEXT	glSelectTextureTransformEXT
glMultiTexCoord2dEXT	glTexCoordPointerEXT
glMultiTexCoord2dvEXT	glTexImage3DEXT
glMultiTexCoord2fEXT	glTexSubImage3DEXT
glMultiTexCoord2fvEXT	glVertexPointerEXT

Mesa specific OpenGL extensions

glResizeBuffersMESA	glWindowPos2svMESA	glWindowPos3svMESA
glWindowPos2dMESA	glWindowPos3dMESA	glWindowPos4dMESA
glWindowPos2dvMESA	glWindowPos3dvMESA	glWindowPos4dvMESA
glWindowPos2fMESA	glWindowPos3fMESA	glWindowPos4fMESA
glWindowPos2fvMESA	glWindowPos3fvMESA	glWindowPos4fvMESA
glWindowPos2iMESA	glWindowPos3iMESA	glWindowPos4iMESA
glWindowPos2ivMESA	glWindowPos3ivMESA	glWindowPos4ivMESA
glWindowPos2sMESA	glWindowPos3sMESA	glWindowPos4sMESA
glWindowPos4svMESA		

SGI OpenGL extensions

glMultiTexCoord1ldSGIS	glMultiTexCoord3fvSGIS
glMultiTexCoord1dvSGIS	glMultiTexCoord3iSGIS
glMultiTexCoord1fSGIS	glMultiTexCoord3ivSGIS
glMultiTexCoord1fvSGIS	glMultiTexCoord3sSGIS
glMultiTexCoord1liSGIS	glMultiTexCoord3svSGIS
glMultiTexCoord1livSGIS	glMultiTexCoord4dSGIS
glMultiTexCoord1lsSGIS	glMultiTexCoord4dvSGIS
glMultiTexCoord1lsvSGIS	glMultiTexCoord4fSGIS
glMultiTexCoord2dSGIS	glMultiTexCoord4fvSGIS
glMultiTexCoord2dvSGIS	glMultiTexCoord4iSGIS
glMultiTexCoord2fSGIS	glMultiTexCoord4ivSGIS
glMultiTexCoord2fvSGIS	glMultiTexCoord4sSGIS
glMultiTexCoord2iSGIS	glMultiTexCoord4svSGIS
glMultiTexCoord2ivSGIS	glMultiTexCoordPointerSGIS
glMultiTexCoord2sSGIS	glSelectTextureCoordSetSGIS
glMultiTexCoord2svSGIS	glSelectTextureSGIS
glMultiTexCoord3dSGIS	
glMultiTexCoord3dvSGIS	
glMultiTexCoord3fSGIS	

8 The libGLU library

Library is not yet implemented.

9 The libGLUT library

Library is not yet implemented.

10 The libTrueType library

Library is not yet implemented.

Index

B

board structure

- pk_create_board 12, 16, 19
- pk_destroy_board 16
- pk_get_board 20, 27
- pk_get_boardinfo 20
- pk_set_framebuffer 20, 21

C

colormap

- pk_allocate_closest_color 25, 26
- pk_allocate_color 25, 26, 27
- pk_allocate_colorcell 25, 26, 27
- pk_allocate_colorcells 25, 26, 27
- pk_allocate_colorplanes 26, 27
- pk_create_colormap 33
- pk_destroy_colormap 33
- pk_get_color 27, 33
- pk_get_colormap 33
- pk_get_colors 27
- pk_get_colors_random 27
- pk_lookup_color 25
- pk_release_color 26
- pk_release_colorplanes 26
- pk_release_colors 26
- pk_set_color 25, 26, 27
- pk_set_colors 26, 27
- pk_set_colors_random 27

cursor

- pk_get_cursordrawable 22
- pk_position_cursor 22
- pk_set_cursorhotspot 22

D

double buffering

- pk_swap_buffers 21, 23

drawable

- pk_back_drawable 28
- pk_create_back_drawable 28
- pk_create_colormap 33
- pk_create_drawable 22
- pk_create_front_drawable 28
- pk_create_overlay_drawable 28
- pk_create_primary_drawable 28
- pk_destroy_colormap 33
- pk_destroy_drawable 22, 29
- pk_destroy_drawable 29
- pk_front_drawable 27
- pk_get_colormap 33
- pk_get_drawable 21

- pk_get_size 29
- pk_overlay_drawable 21, 28
- pk_primary_drawable 28
- drawing methods
 - pk_bitblt 35
 - pk_block_copy 35
 - pk_draw_arc 31
 - pk_draw_char 33
 - pk_draw_circle 31
 - pk_draw_ellipse 31
 - pk_draw_line 30, 31
 - pk_draw_linesegs 31
 - pk_draw_piearc 32
 - pk_draw_pixel 30
 - pk_draw_polygon 31
 - pk_draw_polyline 31
 - pk_draw_rectangle 30, 31
 - pk_draw_triangle 30
 - pk_fill_circle 32
 - pk_fill_ellipse 33
 - pk_fill_piearc 33
 - pk_fill_polygon 32
 - pk_fill_rectangle 32
 - pk_fill_triangle 32
 - pk_read_span 36
 - pk_shaded_line 35
 - pk_shaded_triangle 36
 - pk_write_span 36

E

errors

- pk_get_errorcode 21, 22
- pk_set_errorcode 21

F

font

- pk_create_ti_font 22, 23
- pk_create_ti_font_mem 23
- pk_create_truetype_font 22
- pk_destroy_font 37
- pk_draw_char 33
- pk_draw_string 34
- pk_draw_string16 34
- pk_draw_string32 34
- pk_draw_text 33, 34
- pk_draw_text16 34
- pk_draw_text32 34
- pk_get_ascent 37
- pk_get_descent 37
- pk_get_spacing 37
- pk_glyph_info 37, 38

L

libPK data structures

Pk_board	16
Pk_colormap.....	16
Pk_drawable	16
Pk_font.....	17
Pk_glyph_info.....	17
Pk_pixelinfo	17
libVFX.....	39

P

pan

pk_pan_absolute.....	23, 24
pk_pan_relative	23

pixel format

VFX-M	42
-------------	----

pixmaps

pk_read_pixmap	36
pk_write_pixmap.....	36, 37

pk_allocate_closest_color.....

25, 26

pk_allocate_color.....

25, 26, 27

pk_allocate_colorcell.....

25, 26, 27

pk_allocate_colorcells

25, 26, 27

pk_allocate_colorplanes

26, 27

pk_back_drawable

28

pk_bitblt

35

pk_block_copy.....

35

Pk_board

16

Pk_colormap.....

16

pk_create_back_drawable

28

pk_create_board

12, 16, 19

pk_create_colormap

33

pk_create_drawable

22

pk_create_front_drawable

28

pk_create_overlay_drawable

28

pk_create_primary_drawable

28

pk_create_ti_font.....

22, 23

pk_create_ti_font_mem

23

pk_create_truetype_font

22

pk_destroy_board

16

pk_destroy_colormap

33

pk_destroy_drawable

22, 29

pk_destroy_font

37

pk_draw_arc

31

pk_draw_char.....

33

pk_draw_circle

31

pk_draw_ellipse.....

31

pk_draw_line

30, 31

pk_draw_linesegs

31

pk_draw_piearc

32

pk_draw_pixel.....

30

pk_draw_polygon

31

pk_draw_polyline

31

pk_draw_rectangle

30, 31

pk_draw_string

34

pk_draw_string16	34
pk_draw_string32	34
pk_draw_text	33, 34
pk_draw_text16.....	34
pk_draw_text32.....	34
pk_draw_triangle	30
Pk_drawable	16
pk_extract_color.....	30
pk_fill_circle.....	32
pk_fill_ellipse.....	33
pk_fill_piearc	33
pk_fill_polygon	32
pk_fill_rectangle	32
pk_fill_triangle	32
pk_flush.....	37
Pk_font	17
pk_front_drawable	27
pk_get_ascent	37
pk_get_boardinfo	20
pk_get_cliprect	29
pk_get_color.....	27, 33
pk_get_colormap	33
pk_get_colors	27
pk_get_colors_random	27
pk_get_cursordrawable	22
pk_get_depth	29
pk_get_descent	37
pk_get_drawable	21
pk_get_errorcode	21, 22
pk_get_memspace	29
pk_get_pixelinfo	24
pk_get_pixformat	29
pk_get_size	29
pk_get_spacing	37
pk_get_videosize	21
pk_glyph_info	37, 38
Pk_glyph_info	17, 38
pk_lookup_color	25
pk_make_color	30
pk_overlay_drawable	21, 28
pk_pan_absolute	23, 24
pk_pan_relative	23
Pk_pixelinfo	17, 42
pk_position_cursor	22
pk_primary_drawable	28
pk_read_flash	24
pk_read_pixel	30
pk_read_pixmap	36
pk_read_seeprom	24
pk_read_span	36
pk_release_color	26
pk_release_colorplanes	26
pk_release_colors	26
pk_set_cliprect	29
pk_set_color	25, 26, 27
pk_set_colors	26, 27

pk_set_colors_random.....	27
pk_set_cursorhotspot.....	22
pk_set_errorcode	21
pk_set_framebuffer.....	20, 21
pk_shaded_line	35
pk_shaded_triangle.....	36
pk_swap_buffers.....	21, 23
pk_video_enable	23
pk_write_flash.....	24
pk_writePixmap	36, 37
pk_write_seeprom.....	24
pk_write_span.....	36
programmable memory	
pk_read_flash.....	24
pk_read_seeprom.....	24
pk_write_flash	24
pk_write_seeprom.....	24
V	
VCQ-M.....	46
VFX-M	39